# APRENDER A PROGRAMAR JUGANDO


Trabajo de fin de Grado

**Francisco Faulkner Antiñolo**

Grado en Ingeniería de Computadores
Facultad de Informática
Universidad Complutense de Madrid
Enero 2019

---

Dirigido por:
Alberto de la Encina Vara

# Índice

# Agradecimientos

Este trabajo cierra una etapa muy importante de mi vida. Una etapa llena de positivismo en forma de esfuerzo, ilusión y ganas de aprender cosas nuevas. Aunque en numerosas ocasiones ha sido una cuesta con una gran pendiente muy difícil de subir donde han hecho aparición la frustración y la desesperación. Durante estos años he aprendido muchos conocimientos relacionados con la ingeniería, la informática y la programación. Pero precisamente las dificultades, a priori lo peor de todo, han sido las que me han proporcionado las enseñanzas más valiosas para el resto de mi vida. He aprendido a superarme a mí mismo, a darme cuenta de que soy capaz de sacar adelante lo que me proponga, a ser responsable y ambicioso. Y todo esto no habría sido posible sin las personas que me han acompañado en este camino.

Quiero agradecer en primer lugar la ayuda que me han proporcionado todas las personas con las que he compartido estos años, alumnos y profesores. A mis compañeros con los que he compartido esta etapa y he compartido días de estudio, alegrías y tristezas. A los profesores que me han enseñado los conocimientos para superar cada una de las asignaturas. Y también mencionar a cualquier persona que me ha ofrecido su ayuda en la universidad siempre que lo he necesitado. Sé que toda la ayuda de todas estas personas me ha llevado a donde estoy ahora.

También quiero agradecer a mi familia la oportunidad que me ha proporcionado de poder estudiar un grado universitario. Por su apoyo para que no me rindiera y siguiera adelante sin importar el gasto económico que eso supusiera.

Además, quiero mostrar mi agradecimiento a mi tutor, Alberto de la Encima Vara. Me dio la oportunidad de realizar este Trabajo de Fin de Grado tan interesante y me dejó libertad para realizarlo a mi manera. También quiero agradecer sus palabras en el momento adecuado para afrontar este proyecto con éxito.

Por último, quiero agradecer a todas las personas que, fuera de la facultad, forman o han formado parte de mi entorno. Amigos de toda la vida, conocidos o personas con las que he perdido el contacto o se distanciaron de mí a lo largo de estos años, pero en su día fueron las más importantes. Esas personas que han creído en mí y me enseñaron a perseguir mis sueños.

Muchas gracias a todos.

Francisco.

# Introducción

Hace algunos años, parecía impensable que alguien que no fuera un gran informático pudiera aprender a manifestar sus ideas en lenguaje computacional. Pero en la actualidad, la constante evolución tecnológica ha expandido el uso de código informático a la mayoría de campos profesionales. La programación se ha convertido en una herramienta muy importante en la sociedad y prueba de ello es la aparición de multitud de lenguajes de programación aplicados a numerosos campos, a la vez que progresa el avance tecnológico.
Ahora no solamente generan código los ingenieros e informáticos. Economistas, matemáticos, físicos o geógrafos son ejemplos de profesiones que han ido transformándose para poder satisfacer los nuevos retos que surgen hoy en día. La programación se ha convertido en un instrumento esencial en muchos de los caminos laborales. Pero no exclusivamente en este ámbito. Conscientes de este gran auge, numerosos países han introducido en sus escuelas asignaturas de fundamentos de la informática y la programación para que los niños puedan descubrir todo lo que ofrece este mundo y así poder elaborar sus primeros códigos. La sociedad se adentra cada vez más y más en la era digital. La tecnología avanza a pasos agigantados y con ello el deseo y la curiosidad de muchas personas por este mundo.

Por ello hay una gran cantidad de gente que, a pesar de no dedicarse en su vida profesional a desarrollar código, ha encontrado en la programación un pasatiempo o una manera de dar forma a sus ideas. Existen multitud de posibilidades para empezar a aprender a programar. Tutoriales gratuitos o de pago, videos en plataformas multimedia, cursos online o presenciales, permiten que cualquier persona puede realizar su primer programa en el lenguaje deseado. A su vez, han aparecido páginas web que ofrecen divertidos juegos para que los niños aprendan a programar. Aprovechando que los niños se inician en el uso de aparatos electrónicos a una edad cada más temprana, estos juegos son una opción muy acertada para que los niños desde una tablet, un ordenador o un smartphone, puedan obtener algunos conocimientos sobre este tema. Aprender a dominar un lenguaje de programación está al alcance de todos.
Y a partir de esas ideas surge este proyecto. En este trabajo se quieren desarrollar una serie de juegos que ayuden al jugador a aprender algunos conocimientos de programación. Estos juegos se solucionarán mediante bloques visuales de código que simularán algunos conceptos de programación.

Pero para realizar este proyecto primero hay que hacer un estudio sobre las diferentes herramientas que implementan bloques visuales de código, sobre las distintas maneras que hay de realizar un juego y sobre las distintas webs que ofrecen juegos para aprender a programar. Una vez se haya recopilado toda esta información podremos elegir las herramientas más adecuadas para realizar tanto los juegos como sus bloques de código visuales para crear sus soluciones además de conseguir que ambos interactúen entre sí. Así mismo, en esta memoria se pretende mostrar el proceso de creación de juegos con la finalidad de que el usuario aprenda algunos conocimientos de programación mediante la resolución de los mismos con bloques visuales de programación.

Ya lo dijo Steve Jobs y parece que cada vez más personas son conscientes de ello: "Everybody in this country should learn to program a computer, because it teaches you how to think".

# Summary

A few years ago, it seemed unthinkable that someone, who was not a great computer technician, could learn to express their ideas in computer language. However, currently, the constant technological evolution has expanded the use of computer code in most professional fields. Programming has become a very important tool for society. For instance, the birth of plenty of programming languages which are applied to many fields. At the same time, technological advance is improving.

Nowadays, many professionals, such as economists, mathematicians, physicists or geographers can generate codes, not only enginners and informatics. It means that a lot of professions have been changed in order to satisfy new challenges. Programming has become a very essential tool in many working fields. In addition, plenty of countries have introduced computing and programming subjects in school to teach children how to create new codes. Society makes progress in digital age. Technology comes in leaps and bounds and with that, desire and curiosity of many people in this world. For that reason, there are a lot of people who, despite not dedicating themselves to his professional life to develop code, has found in programming a hobby or a way to shape their ideas. There are many possibilities to start learning to program. Free tutorials or payment, videos on multimedia platforms, online or face-to-face courses allow anyone to make their first program in the language that he wanted. In turn, websites that offer fun games have appeared to children can learn to program. Taking advantage of the fact that children start using electronic devices at an earlier age, these games are a very successful option for them to obtain some knowledge about this matter from a tablet, a computer or a smartphone. Learning to master a programming language is within reach of all.

From these ideas comes this project. In this work they want develop a series of games that help the player to learn some programming knowledge. These games will be solved by visual blocks of code that will simulate some programming concepts. However, in order to carry out this project, it must be made a study about the different tools that implement visual blocks of code, about the different ways that there are to realize a game and the different webs which offer games to learn to program. Once it has been collected all this information, it will be possible to choose the most appropriate tools

for perform both the games, their visual code blocks to create their solutions and getting them both to interact with each other. Likewise, in this memory it is intended to show the process of creating games with the purpose of learning some programming knowledge by solving them with visual blocks of programming.

Steve Jobs said and it already and it seems that more and more people are aware of it: Everybody in this country should learn to program a computer, because it teaches you how to think.

# 1. Motivación y objetivos

## 1.1. Motivación

Como se ha mencionado, cuando se trata de aprender a programar existen multitud de aplicaciones y sitios web dedicados a ello. En los últimos tiempos, han tomado fuerza las webs que enseñan conceptos básicos de programación mediante sencillos juegos, hasta tal punto en el que muchas escuelas de diversos países las utilizan para impartir docencia. Principalmente enfocadas a niños de determinadas edades, estos juegos utilizan la programación con bloques que sustituyen a las sentencias de código. De esta manera, como si de un rompecabezas se tratara, los más pequeños y no tan pequeños pueden resolver los retos que se presentan a la vez que reciben algunos conocimientos sobre la lógica de la programación. ¿Pero podemos ser capaces de hacer fácilmente nuestro propio juego cuya solución nos aporte conocimientos de programación? La respuesta es sí y existen multitudes de tecnologías para realizarlo.

## 1.2. Objetivos

En primer lugar, se van a conocer algunas de las webs que enseñan a programar mediante juegos y se hará un breve análisis de cada una. También se darán a conocer algunos frameworks que se utilizan para desarrollar juegos y algunas de las herramientas que permiten utilizar bloques visuales de programación. Una vez realizados estos dos análisis, se escogerán dos tecnologías, Phaser y Google Blockly. Una vez se ha aprendido a utilizar estas dos tecnologías, se puede realizar el objetivo principal del proyecto. Este objetivo principal consiste en la elaboración de juegos en HTML y Javascript en cuya resolución hay que emplear bloques visuales de código. La finalidad de estos juegos será que el usuario reciba conocimientos de programación. A su vez también se plantea como objetivo que esta memoria sirva de guía para que el usuario pueda ser capaz de aprender a realizar sus propios juegos con ayuda de Phaser, de entender la lógica de bloques, construir sus propios bloques con Google Blockly y finalmente, realizar sus propios juegos que incluyan un panel de bloques visuales para su resolución.

Tareas realizadas en el proyecto:

1. Estudio de las principales webs que enseñan a programar mediante juegos.

2. Estudio de las principales herramientas que nos permiten utilizar bloques visuales de código.

3. Estudio de los principales frameworks que sirven para desarrollar juegos.

4. Aprendizaje de Google Blockly, herramienta de Google que permite utilizar bloques visuales de código.

5. Aplicación e inserción de un panel de bloques en una web.

6. Aprendizaje sobre el manejo de Phaser, un framework que facilita la elaboración de juegos mediante Javascript y HTML. Se enseñan sus funcionalidades básicas para poder crear un juego.

7. Elaboración de juegos que transmiten conocimientos de programación uniendo ambas tecnologías. Se utilizará Phaser para el diseño y Google Blockly para poder llevar a cabo la solución del juego mediante bloques visuales de código.

8. Elaboración de una memoria mediante lenguaje LaTeX que contenga los apartados anteriormente enumerados.

# Motivation and goals

## 1.3. Motivation

As mentioned previously, when it comes to learning to program, there are multitude of applications and websites which are dedicated to that. In the last times, webs that teach basic programming concepts have gained strength through simple games, to such an extent that many schools in various countries use them to teach. Mainly, these webs have been focused on children of certain ages and these games use programming with blocks that replace the code sentences. In this way, as a puzzle, young kids and other people can solve the challenges that arise while they receive some knowledge about the logic of programming. But, can we be able to do our own game easily whose solution gives us knowledge of programming? The answer is yes and there are a lot of technologies to perform it.

## 1.4. Goals

Firstly, it will be known some of the websites that teach program through games and a brief analysis of each will be done. Secondly, it will be announced some frameworks that are used to develop games and some of the tools that allow to use visual blocks of programming. Once these two analyzes are done, two technologies, Phaser and Google Blockly, will be chosen. After learning the use of these two technologies, the main objective of the project can be realized. The main objective consists of the elaboration of HTML and Javascript games in which resolution you have to use visual blocks of code. The purpose of these games will be that the user receives programming knowledge. At the same time, other objective will be also that this report serves as a guide so that the user can be able to learn how to make their own games with help of Phaser, to understand the logic of blocks, to build their own blocks with Google Blockly and finally, make your own games that include a panel of visual blocks for resolution.

Tasks carried out in the project:

1. A Study of the main websites that teach programming through games.

2. A Study of the main tools that allow us to use visual blocks of code.
3. A Study of the main frameworks that serve to develop games.
4. . Learning from Google Blockly, a Google tool that allows use visual blocks of code.
5. Application and insertion of a panel of blocks in a web.
6. Learning about the handling of Phaser, a framework that facilitates the development of games using Javascript and HTML. They teach their basic functionalities to be able to create a game.
7. Development of games that transmit programming knowledge uniting both technologies. Phaser will be used for the design and Google Blockly to be able to carry out the solution of the game by means of visuals blocks code
8. Preparation of a memory using LaTeX language containing the sections previously listed.

## Contenido de la memoria

La memoria comienza con una primera parte donde se resumen los temas principales de los que se va a hablar en este proyecto. En primer lugar, se presenta una introducción que pone en contexto la temática de este trabajo, se exponen las motivaciones que han llevado a realizarlo, se muestran los objetivos que se quieren lograr y los conocimientos que se quieren transmitir al receptor. También se listarán las herramientas software que se han utilizado en este proyecto.

En el siguiente capítulo se realiza un recorrido por las páginas webs más representativas que contienen juegos con los que el usuario puede recibir conocimientos sobre programación. De cada una se muestra una breve descripción, funcionamiento, diferencias y virtudes.

En el tercer capítulo se darán a conocer algunas herramientas que nos permiten utilizar bloques visuales de código. Se analizarán sus ventajas e inconvenientes para así poder seleccionar la que más se adapta al proyecto que se quiere realizar.

El cuarto capítulo hablará exclusivamente de Google Blockly, la tecnología de bloques visuales que hemos elegido para realizar este proyecto. Se enseñan todas sus posibilidades y lo necesario para implementar esta tecnología en un sitio web. Además, se explicarán dos maneras diferentes de crear nuevos bloques de código: codificándolos con Javascript y fabricándolos en la Block Factory.

Necesitamos encontrar la mejor herramienta para poder realizar un juego. Actualmente existen diversos Frameworks en distintos lenguajes de programación que ayudan al usuario a realizar el diseño de su juego fácilmente. Con ello se consigue que toda la atención se centre en la lógica del juego. Es por ello que, en este quinto capítulo, se realizará un breve análisis de algunos de los Frameworks más famosos para realizar juegos con HTML y Javascript. Este capítulo ayudará a decantarnos por uno de los frameworks para realizar los juegos de este proyecto.

Una vez hemos conocido diferentes Frameworks para realizar juegos en HTML, se escogerá Phaser para diseñar los juegos de este proyecto. El sexto capítulo

será una guía sobre el uso de Phaser. El objetivo es que, tras leer este capítulo, el usuario sea capaz de comenzar a crear su propio juego.

En este séptimo capítulo se explica el proceso completo para realizar un juego cuya solución se realice mediante bloques visuales de código. Por ello habrá que unir los juegos realizados en Phaser con los bloques creados con Google Blockly.

En el octavo capítulo de esta memoria se enseñarán los juegos que se han realizado para este proyecto. Se explicará el concepto del juego, los conocimientos de programación que se quieren simular en él y las validaciones realizadas en cada juego.

La memoria termina recogiendo las conclusiones sobre el proyecto realizado. Un resumen general de lo aprendido gracias a la elaboración del trabajo, las dificultades encontradas durante su desarrollo y algunas ideas de trabajo futuro que se podrían realizar a partir de este proyecto, así como la bibliografía utilizada.

# Herramientas utilizadas en este trabajo

Este trabajo está formado por dos partes, la memoria y la aplicación. Esta memoria se ha realizado mediante el lenguaje LaTeX en el editor TeXworks. La aplicación se ha programado con ayuda del editor de código Brackets. A continuación, se detallan las ventajas de estas dos herramientas y el por qué se han utilizado para realizar este trabajo frente a otras:

## TeXworks

Es un editor LaTeX [9] [10] [11] de código abierto. LaTeX es un lenguaje que permite la construcción de textos. Está basado en TeXshop, otro editor LaTeX, pero TeXworks es más simple.
Ventajas:

- Gratuito.
- Código abierto.
- Multiplataforma
- Compatible con Unicode.
- Autocompletado.
- Permite visualizar el PDF resultando y se actualiza en tiempo real mientras se edita el código.
- Interfaz simple y con funcionalidades básicas.

### Brackets

Es un editor de código [12] [13] para trabajar con lenguajes web como son HTML, CSS y Javascript. Está desarrollado por Adobe Systems bajo la licencia MIT.
Ventajas:

- Código abierto.
- Gratuito.
- Permite visualizar el resultado en tiempo real mientras se edita el código.
- Multiplataforma.
- Soporta múltiples lenguajes de programación.
- Multilenguaje.
- Gran cantidad de plugins que aumentan sus funcionalidades.
- Simplicidad.
- Múltiples variantes para personalizar el entorno.

# 2. Webs y aplicaciones que contienen juegos para aprender a programar

El objetivo de este proyecto consiste en elaborar una serie de juegos que transmitan nociones de programación. Antes de realizarlos y para tener ideas sobre lo que ya existe en el mundo real, se van a analizar algunos de los juegos de este tipo ya existentes. En este capítulo [1] se van a presentar distintas webs que contienen juegos que permiten al usuario aprender a programar. Estas webs son Code.org, CodeMonkey, Code For Life, Codecombat, Check iO, Kodu, Robogarden y Tynker.

## 2.1. Code.org

Tal y como se ha comentado en la introducción de esta memoria, en los tiempos actuales la programación es un pilar fundamental en numerosos trabajos de nuestra sociedad. Cada vez existen más campos profesionales en los que se demandan conocimientos mínimos de programación. Se prevé que al menos en Estados Unidos los puestos de empleo relacionados con programación se extenderán a un millón y medio. Es por ello que surge la idea de que cualquier persona debería aprender a programar. Una buena solución a ello sería impartir estas enseñanzas como asignatura en los colegios o institutos de todo el mundo. A partir de esta idea, nace en Estados Unidos la organización no gubernamental Code de la mano de los hermanos iraníes Ali Partovi y Hadi Partovi. Code ha crecido inmensamente a lo largo de los años y cuenta con un gran número de estrellas para promocionarlo: desde grandes nombres de la informática como son Bill Gates o Mark Zuckerberg, músicos como Enrique Iglesias o Will.i.am, deportistas como la tenista Serena Williams o el jugador de la NBA Chris Bosh, hasta importantes ilustres como expresidente de Estados Unidos Barack Obama o el científico Stephen Hawkings Todo este gran plantel tiene el objetivo de acercar la programación al mayor número de personas posibles mediante diversas plataformas. Y una de estas plataformas es Code.org [25].

Code.org es un sitio web que enseña a programar a través de sencillos de juegos. Es una de las webs más completas de este tipo pues contiene un extenso catálogo de cursos, videos y retos en forma de juegos para diversas edades y niveles. Está orientada a la enseñanza de la programación y por

tanto, para que pueda ser utilizada tanto por educadores, como por alumnos, como por personas anónimas. Los retos y cursos de Code.org se pueden utilizar sin necesidad de registro pero si se quiere tener un seguimiento de la propia cuenta o de un alumno se debe crear una cuenta. Existen dos tipos de cuentas: alumno y profesor. La finalidad del tipo de cuenta profesor es poder crear grupos de alumnos y asignarles uno o más cursos. De esta manera, profesores (o padres) tienen la posibilidad de realizar un seguimiento de cada uno de los alumnos para observar su progresión. La cuenta de tipo alumno sirve para saber cual fue el último reto o curso que realizamos o llevar un seguimiento del progreso personal. Code.org utiliza bloques visuales de Google Blockly para codificar la solución de los distintos juegos de los módulos que constituyen cada curso (de forma muy similar al objetivo que se pretende con este trabajo de fin de grado). Estos juegos son muy atractivos para el público más joven, pues Code.org cuenta con licencias muy aclamadas como Minecraft, Plantas vs Zombies o Frozen de Disney. También permite crear nuestros propios juegos y aplicaciones utilizando los bloques visuales de código. En el capítulo 3, sección 3.5 se aportará más información sobre el editor de Code.org.

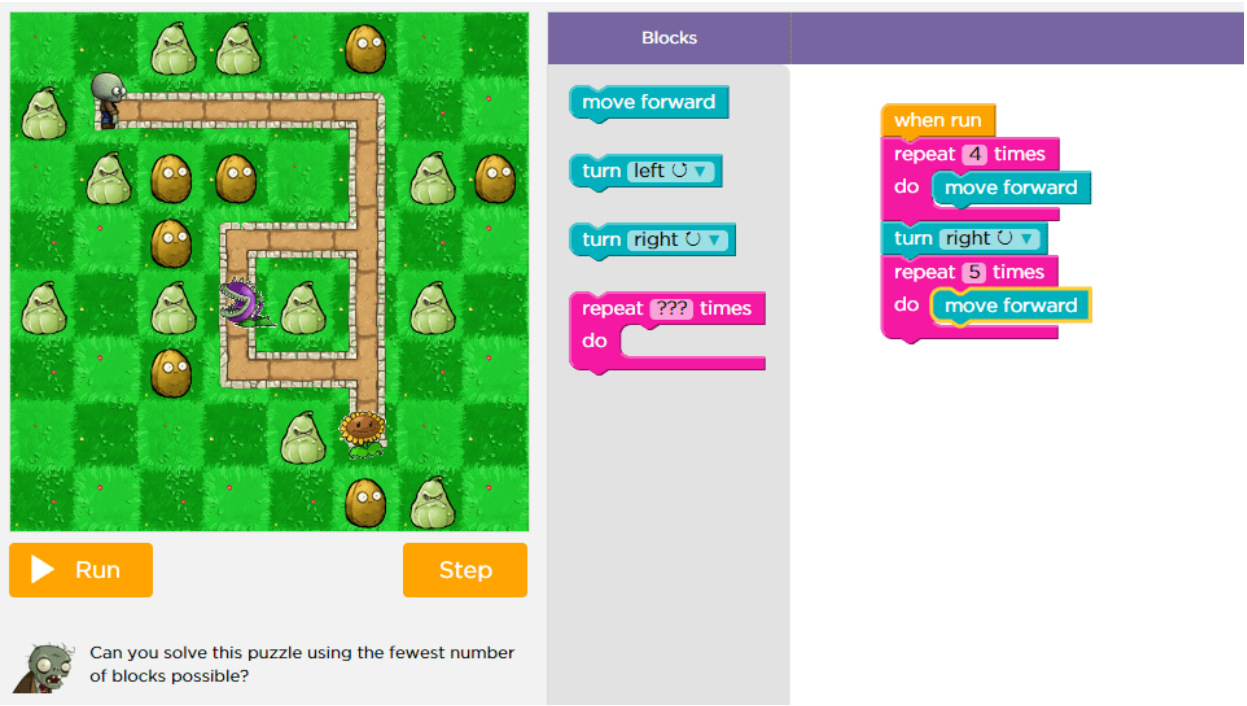


Figura 1: Ejemplo de nivel de Codeorg. Imagen descargada de http://i0.wp.com/robwirving.com

## 2.2. CodeMonkey

Como si de un juego de Code.org se tratara, CodeMonkey [2] nos presenta una interesante manera de aprender a programar desde un navegador web. Un simpático mono será el protagonista de numerosos niveles que el usuario deberá ir solucionando. El objetivo de cada nivel será recoger todos los plátanos del mapa, desplazándonos hacia ellos, cruzando puentes o ríos sobre una tortuga a la que también habrá que programar sus movimientos mediante simples instrucciones. En cada nivel, se mostrará abajo las posibles acciones a realizar, pero no será mediante bloques visuales, sino mediante código escrito. Por ejemplo, en el primer nivel, el más sencillo de todos, se muestra al personaje y al plátano a recoger a una cierta distancia. El usuario en este caso, tendrá que escribir la instrucción *step* y el número de pasos a dar hasta el objetivo para completar el nivel. En niveles más avanzados, habrá que escribir instrucciones como *turn left*, *turn right*(para girar a la

izquierda o a la derecha) o *turtle.step* para mover la tortuga y que así pueda llevar al protagonista a su objetivo. CodeMonkey es una de las herramientas para aprender a programar algo más diferente, pues se aleja de la idea de la programación por bloques, obligando al usuario a tener que escribir sencillas líneas de código. Pero su mayor desventaja es que no es gratuito pues cuesta alrededor de 30 dólares al año.


Figura 2: Ejemplo de nivel de CodeMonkey. Imagen descargada de https://angel.co/codemonkey-studios/jobs

## 2.3. Code For Life

Code For Life [27] es una iniciativa sin ánimo de lucro creada en el año 2014 en Reino Unido por los voluntarios de Ocado Technology. Su objetivo consiste en apoyar a maestros y estudiantes para llevar la programación al mayor número posible de personas. La idea surgió tras la aparición de un nuevo plan de estudios que contenía ciencias informáticas. La mayoría de los

institutos de Reino Unido no tenían herramientas suficientes para proporcionar la enseñanza de programación a sus alumnos, y eso pretende ser Code For Life. Una herramienta que permita a los profesores enseñar fácilmente y a los alumnos aprender de forma divertida. Se caracteriza por su código abierto. Cualquier persona que quiera contribuir voluntariamente puede hacerlo de diferentes maneras. La primera es que cualquiera que haya utilizado su web envíe comentarios sobre Code For Life, pues a sus desarrolladores les interesa bastante las opiniones de sus usuarios. La segunda manera de aportar al proyecto es desde su Github. Cualquier desarrollador puede intentar implementar nuevas funciones o únicamente proponerlas para que otros las puedan llevar a cabo. Esa es la esencia de Code For Life pues cada tipo de persona, sea programador, alumno o profesor puede formar parte de este proyecto de una u otra manera.

En su web podemos encontrar juegos cuya resolución requiere utilizar bloques visuales Blockly. Al igual que en Code.org existen cuentas para profesores y para alumnos. Para los alumnos se proporcionan juegos de aprendizaje. Actualmente solamente existe el juego Rapid Router, en el que debemos guiar a un camión a lo largo de una carretera hasta su destino. Para ello habrá que escoger los bloques correctos. Pero no solamente hay bloques para mover el camión. También es necesario utilizar bucles y condicionales, por ejemplo, para poder parar el camión mientras el semáforo esté en rojo, habrá que utilizar un bucle *while* para parar el camión hasta que el semáforo se ponga en verde y permita el paso. En los niveles más avanzados se encuentra una de las mejores características que posee esta web. Al posicionar los bloques que se quieren utilizar sobre el panel de bloques se muestra la traducción en código Python para que el alumno aprenda que hacen realmente dichos bloques. Realmente con ello se garantiza que el alumno acabará los cursos de la web con algunas nociones de Python y así poder dar el salto a la codificación. La web también contiene un editor de niveles sencillo de utilizar, pero con muchas opciones que permiten crear un recorrido propio para el camión pudiendo diseñar la carretera, los obstáculos y los bloques Blockly que se deben utilizar para resolverlo. Para las cuentas de profesores, se pueden organizar grupos de alumnos para hacer un seguimiento, ver sus puntuaciones e incluso existen vídeos para guiar al mentor y proporcionarle un modelo de enseñanza para sus pupilos. Además de cara al futuro, se está implementando un nuevo juego llamado AI:MMO. Este juego destinado a niños de edades comprendidas entre 13 y 16 años, será un juego multijugador en línea orientado a

enseñar conocimientos de Python e Inteligencia Artificial.


Figura 3: Ejemplo de nivel de Code For Life. Imagen propia

## 2.4. Codecombat

Una de las opciones más completas y atractivas para aprender a programar. Codecombat [28] nos presenta un interesante juego de rol realizado en HTML5 y CoffeeScript. Está formado por numerosos niveles y en cada uno de ellos se aprenderá y se utilizará código Python y código Javascript. Permite elegir un héroe entre los 16 que se ofrecen. Cada uno de los personajes tiene su propio tipo como puede ser guerrero a corta distancia, a larga distancia o mago, sus propias estadísticas de ataque, salud y velocidad y sus propias armas y habilidades. Además, la dificultad del juego será más fácil o difícil según el protagonista que se haya escogido. Algunos héroes no son gratuitos y requieren una suscripción mensual por 10 dólares o una única suscripción de 100 dólares para poder usarlos siempre. Una vez elegido, antes de cada nivel podremos personalizarlo con ropa y objetos que iremos consiguiendo a lo largo del juego. Estos objetos proporcionan puntos extra de ataque, salud o velocidad, pero también nuevas acciones (que se traducen en una función de programación asociada al personaje). También podremos comprar algunos objetos pagando con las gemas que conseguiremos a lo largo del juego. Cada uno de los desafíos plantea un punto origen desde el cual habrá que llegar al

destino. Por el camino habrá diferentes obstáculos y enemigos que dificultaran la tarea. Para ello habrá que utilizar las acciones de nuestro personaje que no son otra cosa que funciones que se ejecutan sobre él. A disposición se tiene un editor de código en el que se muestran las acciones posibles a realizar, como moverse o atacar para que el usuario las introduzca por teclado. Para los más inexpertos tendremos opciones de ayuda y autocompletado. Una vez completado el código se podrá ejecutar para ver si con esas acciones el personaje llega a su destino y así se completa el nivel. Si no, se podrán realizar más intentos hasta conseguirlo. Al completar el nivel, ganaremos puntos de experiencia, gemas y objetos. También dispone de algunos desafíos online contra otros jugadores donde se compite por completar el nivel antes que nuestros adversarios. Un juego muy completo y trabajado. Sin duda es una de las opciones más entretenidas para aprender a programar.


Figura 4: Ejemplo de nivel de Codecombat. Imagen descargada de https://discourse.codecombat.com/t/kith-an-ice-guy/11602

## 2.5. Check iO

Alexander Lyabah es un desarrollador ucraniano que ha creado esta web que, a pesar de no ser tan conocida como las citadas anteriormente, sí que es una de las que mayor curva de aprendizaje ofrece. Esta web [30] permite aprender Python y Javascript mediante los retos que plantea. A través de un mundo compuesto por númerosas islas, cada uno de ellas ofrecerá uno o varios desafíos. Eso sí, es necesario que el usuario tenga conocimientos previos de programación ya que la resolución de los juegos no se realiza mediante bloques visuales sino mediante líneas de código. Antes de cada uno de los retos, se nos presentará una extensa introducción en la que se proporcionarán algunos conocimientos y elementos básicos para poder resolverlos, así como la precondición que debe cumplir la solución. También se pueden visualizar las estadísticas para saber cuántas islas se han completado y cuantas quedan por resolver. Actualmente cuenta con una extensión para el navegador Mozilla Firefox y así poder ser usado más fácilmente. Como punto negativo encontramos algunos retos más avanzados que desde hace unos años dejaron de ser gratuitos.

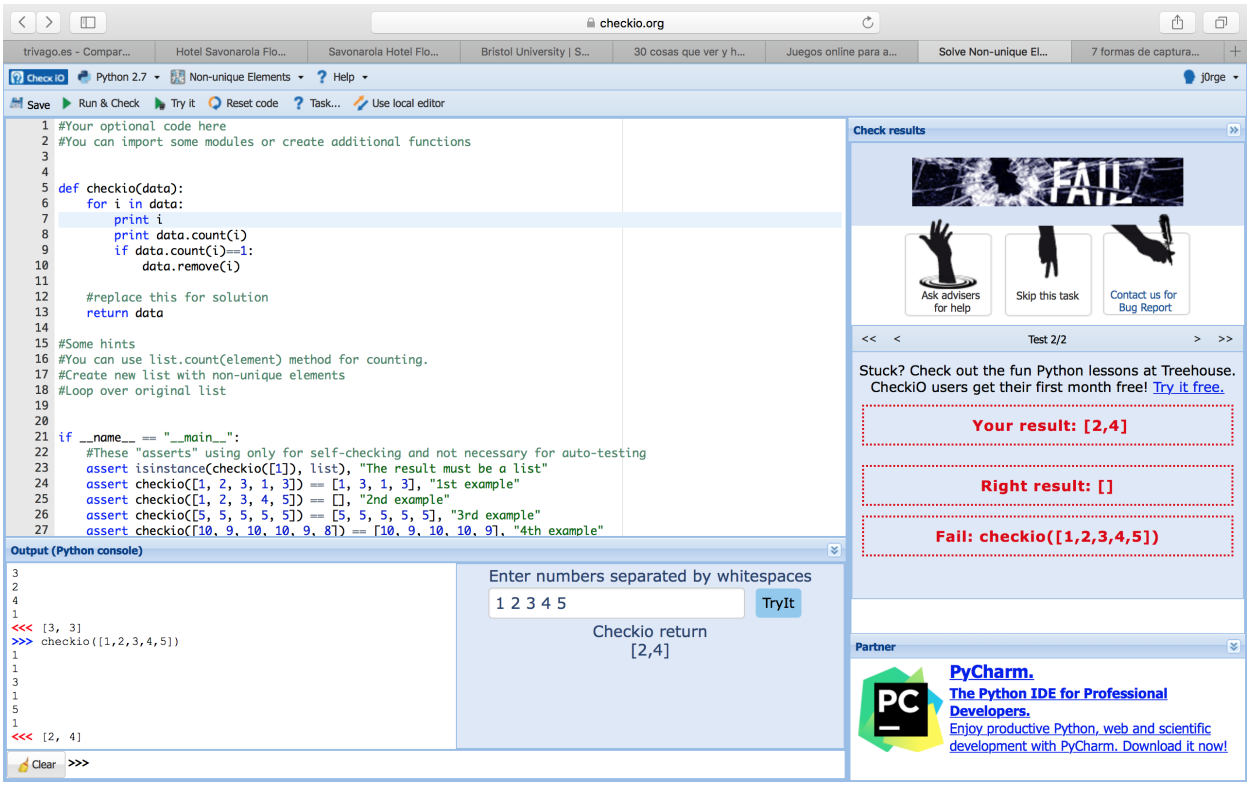


Figura 5: Check iO. Imagen descargada de https://checkio.org/

## 2.6. Kodu

Kodu [7] [8] es un lenguaje de programación visual que se utiliza para crear videojuegos. Como la mayoría de lenguajes de este tipo, no requiere conocimientos previos de lógica de programación. Orientado a niños de edades escolares, Kodu les permitirá crear su propio videojuego como si de uno más se tratase. Una vez descarga e instalada la aplicación de escritorio, nos permitirá construir mundos en tres dimensiones a través de su sencilla interfaz. Inicialmente se nos proporcionará un mundo vacío sobre el que el usuario debe ir construyendo y programando el comportamiento de los distintos elementos que lo formarán Kodu también cuenta con clases y ejemplos que muestran y enseñan los distintos pasos a seguir para construir un mundo propio. A través de su interfaz de botones se pueden ir seleccionando las diferentes opciones para completar cada mundo introduciendo al usuario al-

gunos conceptos de lógica, matemáticas y programación. Por ejemplo, con uno de los botones se agregan objetos al entorno y con otro de ellos, definir el comportamiento con bloques de programación visual. La ventaja principal de Kodu es la posibilidad de poder obtener conocimientos de programación en un entorno divertido y tridimensional, a través de sus ejemplos y lecciones. Como desventaja se tiene que los tipos de objetos a programar y el entorno son algo limitados.


Figura 6: Kodu. Imagen descargada de https://tusejemplos.com/ejemplos-de-kodu-lab/

## 2.7. RoboGarden

Robogarden [26] es una web que contiene juegos para enseñar a programar a niños de diferentes edades. En los diferentes retos se controla a un robot que deberemos guiar por el mapa con ayuda de bloques visuales de código de Google Blockly, muy similar a lo que se quiere realizar en este proyecto y a los juegos de otras webs como Code.org. Todo ello con unos gráficos de bastante nivel y un motor en tres dimensiones. RoboGarden también dispone de niveles más avanzados para resolver con código Javascript sin necesidad de bloques visuales de código. En total dispones de 680 misiones repartidas en ocho mundos. Se comienza desde los niveles más sencillos en los que únicamente hay que mover al robot hacia delante y hacia atrás y según se van superando esos niveles se presentan otros más complejos donde ya habrá que utilizar bucles o variables condicionales. Existen diferentes tipos de cuenta para estudiantes, profesores o escuelas. Los niveles más básicos se pueden jugar libremente sin previo registro, pero para continuar con los demás hay que realizar un registro de usuario gratuito. Las cuentas de profesores o escuelas permiten agrupar alumnos previamente registrados y realizar un seguimiento grupal o individual. Por tanto, RoboGarden es otra de las webs que apuestan por integrar este modo de transmitir conocimientos de programación en el sistema educativo, proporcionando una herramienta de enseñanza para los profesores. La web está disponible en español, portugués, inglés y árabe.


Figura 7: Robogarden. Imagen descargada de https://robogarden.ca/es

## 2.8. Tynker

Tynker [29] presenta diferentes retos y puzzles para solucionar con bloques de programación visuales, similar a Code.org o Code For Life. También permite utilizar para usuarios más experimentados líneas de código en Javascript, Python o HTML en lugar de los bloques visuales de código. Dispone de algunas licencias infantiles como Barbie y HotWheels. Cuenta con varios tipos de registro. Un usuario puede registrarse como alumno, como profesor para tener una cuenta necesaria que le proporcione todo lo necesario para enseñar y monitorizar a sus alumnos y por último tenemos un tipo de usuario padre que sirve para poder observar las creaciones de la cuenta que tengamos asignada como hijo. Todas estas cuentas proporcionan las funcionalidades básicas gratuitas, pero si se quiere acceder a todos los servicios de Tynker hay que suscribirse previo pago. Los precios son 20 dólares por un único mes, 120 dólares por un año o 240 dólares por un servicio de por vida. Además, Tynker permite construir nuestro propia aplicación o juego mediante los bloques visuales de programación. De esta faceta más creativa de Tynker hablaremos en el capítulo 3, sección 3.4.

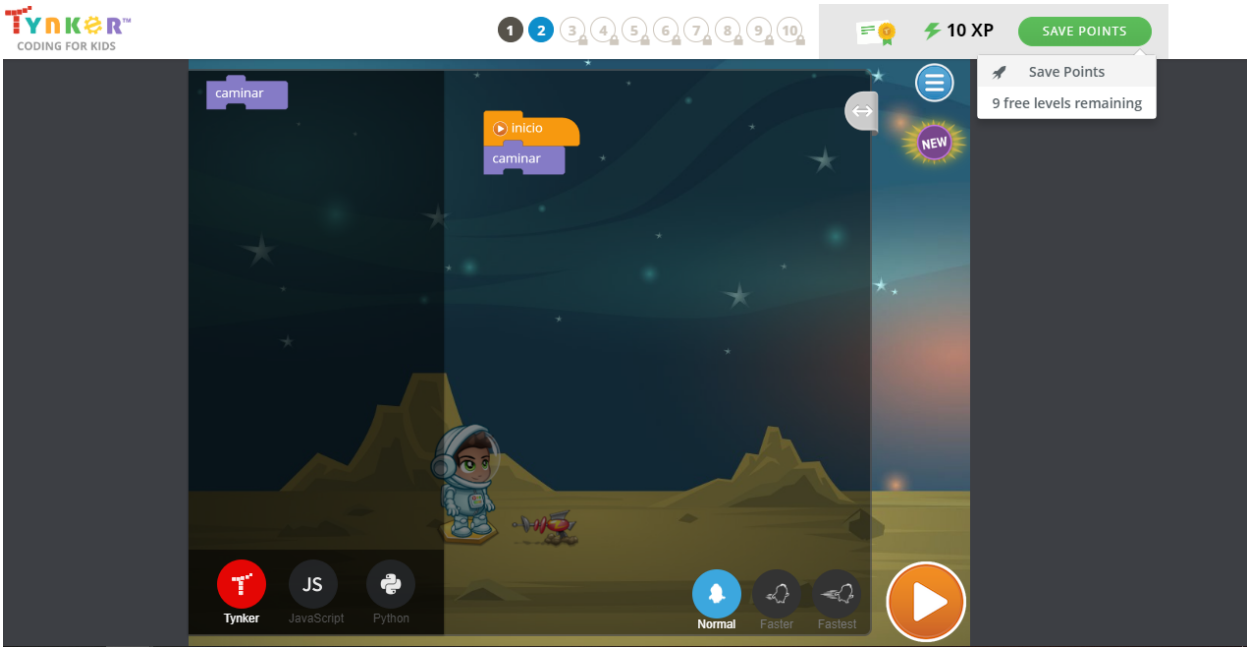


Figura 8: Tynker. Imagen descargada de https://www.tynker.com

## 2.9. Tabla comparativa. Ventajas y desventajas

En la tabla 1 se van a resumir las virtudes y defectos entre las webs presentadas. En las filas se muestra cada web y en las columnas el nombre, sus virtudes y sus defectos. Desde mi punto de vista me ha gustado mucho recopilar información sobre estas webs porque me ha parecido una manera muy interesante de que los usuarios puedan obtener conocimientos de programación. De hecho, es muy probable que utilice alguna de ellas en un futuro como CheckiO o Codecombat. La web más completa y mejor organizada para mí es Code.org. Cuenta con un editor para poder realizar creaciones propias, retos para solucionar mediante Blockly, cuentas tipo profesor y alumno además, de licencias muy atractivas. También me parece la más apta para los más jóvenes y principiantes. Para usuarios que tengan conocimientos de programación, CodeMonkey, CheckiO y Codecombat me parecen las más didácticas pues no utilizan bloques visuales, sino que hay que tener una fluidez a la hora de editar código en los retos más avanzados. En cuanto a nivel de diversión, las webs que más me han entretenido a la hora de realizar este recorrido han sido sin duda Robogarden y Codecombat. De Robogarden me impresionaron

mucho sus gráficos en tres dimensiones y su interfaz. De Codecombat me ha gustado todo pues, es un juego mi completo y personalizable y a mí realmente me entretuvo tanto que solucioné bastantes niveles y seguiré haciéndolo en un futuro. Además, al utilizar código directamente y ser tan entretenido me parece muy adecuado para gente que como yo, está realizando estudios de programación.

| Nombre | Virtudes | Defectos |
|---|---|---|
| CodeMonkey | Aprendizaje directo pues se utilizan líneas código para la resolución de los juegos. | Poco apto para novatos pues no hay bloques de código. No es gratuito. |
| Code.org | Multitud de retos para solucionar con Blockly. Varios tipos de cuentas. Licencias muy atractivas para los más jóvenes. Editor propio. | Aparantemente no se pueden exportar todos los tipos de creaciones realizadas por el usuario. |
| Code For Life | Aprendizaje mediante Blockly pero al terminar los juegos muestra su traducción Python. Editor propio. Varios tipos de cuentas. Cualquier persona puede aportar al proyecto. | Actualmente solo hay un tipo de juego disponible. |
| Codecombat | Uso de código Javascript y Python. Un completo y verdadero juego de rol, subida de niveles, personalización... | Algunos contenidos son de pago, aunque son complementarios. No utiliza bloques visuales de código. |
| CheckiO | Uso de código Javascript y Python. Es el juego que más cerca está de enfrentar al usuario con un código de programación real. Buen seguimiento de estadísticas por parte del usuario. | No utiliza bloques visuales de código. Se necesitan conocimientos previos de programación. No todos los niveles son gratuitos. |
| Kodu | Entorno tridimensional. Interfaz sencilla para los usuarios más jóvenes. | Requiere instalación previa. Entorno muy limitado. |
| Robogarden | Uso de Google Blockly. 680 retos. Entorno tridimensional muy atractivo. | Requiere instalación previa. |
| Tynker | Permite usar tanto bloques visuales como código Javascript, Python o HTML. Múltiples tipos de cuenta. Editor propio | Contenidos de pago. |

Tabla 1: Comparación de Frameworks para crear juegos.

# 3. Lenguajes y herramientas que utilizan bloques visuales de programación

En el apartado anterior se ha realizado un análisis sobre las webs que contienen juegos para aprender a programar. Algunas de ellas utilizan bloques visuales de código. En este apartado [25] [20] vamos a hacer un análisis de varias herramientas que nos permiten desarrollar nuevas aplicaciones con esos bloques de programación y no solamente solucionar juegos ya creados por la propia plataforma.

## 3.1. Scratch

El grupo Lifelong Kindergarten de MIT fue el responsable de Scratch [5] [6], uno de los primeros pioneros en utilizar la programación visual por bloques. Su versión oficial apareció en 2005 tras varias versiones de test desde el año 2002. En 2015 se lanzó una actualización llamada Scratch 2.0. y en agosto de 2018 salió la última versión existente, Scratch 3.0. Es tan popular que se utiliza en numerosos cursos y escuelas de todo el mundo como herramienta de aprendizaje. Su objetivo consiste en poder crear animaciones y juegos mediante una interfaz sencilla, pero a la vez completa, sin que el usuario tenga previamente conocimientos en programación. Para ello se nos proporcionan una serie de elementos a los que dotaremos de acciones o movimientos mediante los bloques. Como si de un rompecabezas se tratara, se irán concatenando los bloques de acciones para obtener el comportamiento deseado de una manera similar a lo visto en Google Blockly. Estos bloques están divididos en varias secciones:

- Movimiento: permite desplazar un elemento sobre la pantalla.
- Apariencia: permite cambiar las propiedades de los elementos como su tamaño o color.
- Sonido: con este tipo de bloques podremos reproducir audios.

- Lápiz: permite dibujar sobre la pantalla.

- Datos: permite declarar variables y darles valor.

- Eventos: los eventos permiten lanzar una determinada acción sobre un elemento o el entorno.

- Control: permite utilizar simplificadamente los típicos bucles que se encuentran en cualquier lenguaje de programación como los bucles If o los bucles For.

- Operadores: permiten realizar operaciones lógicas y matemáticas.

- Sensores: se utilizan para que los elemento interactúen entre ellos o con el entorno.

  La mayor ventaja que presenta Scratch es la posibilidad de compartir las creaciones vía Web en su extensa comunidad virtual y descargar los proyectos de otras personas para modificarlos o ampliarlos. Cualquiera puede escoger un proyecto publicado en su web y abrirlo en su propia interfaz para poder tener su código de bloques. Otra ventaja es la posibilidad de utilizarlo desde la aplicación de escritorio o desde el navegador mediante su web. Además, es software libre y válido para todos los sistemas operativos.

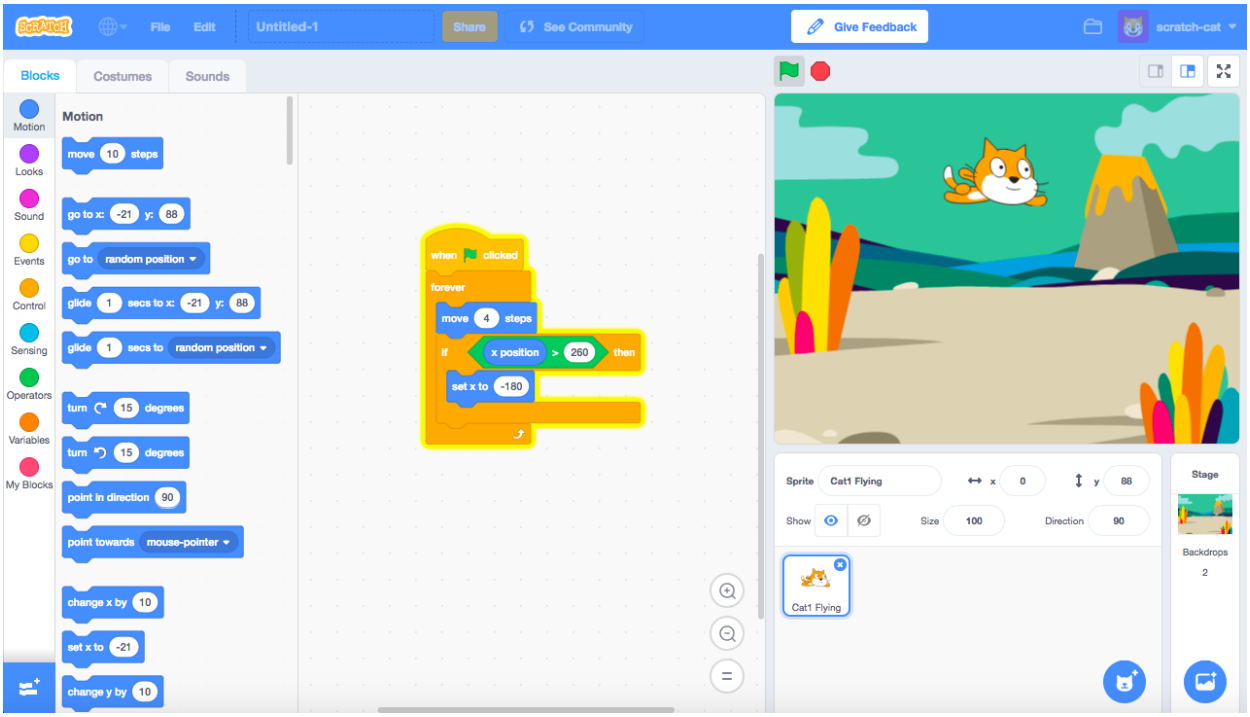


Figura 9: Scratch. Imagen descargada de https://www.paviles.net

## 3.2. Snap, el legado de Scratch

El legado que ha dejado Scratch ha sido muy significativo y prueba de ello son las numerosas herramientas que han surgido derivadas de él, extendiendo algunas de sus funcionalidades. Uno de los sucesores de Scratch más conocidos es Snap [3] [4]. Snap implementa la mayoría de funcionalidades de su progenitor y proporciona algunas mejoras. Está creado con Javascript y HTML5 para ser compatible con todo tipo de navegadores por lo que es más versátil. Además, se ha mejorado la capacidad para crear bloques nuevos, siendo una mejor opción que Scratch si lo que se quiere es añadir funcionalidades nuevas a los bloques visuales. Otra mejora es la posibilidad de agrupar variables por listas y así poder darles un comportamiento en conjunto. Por ello Snap es más recomendado para usuarios que ya tienen una cierta experiencia en Scratch y quieren crear proyectos más extensos.

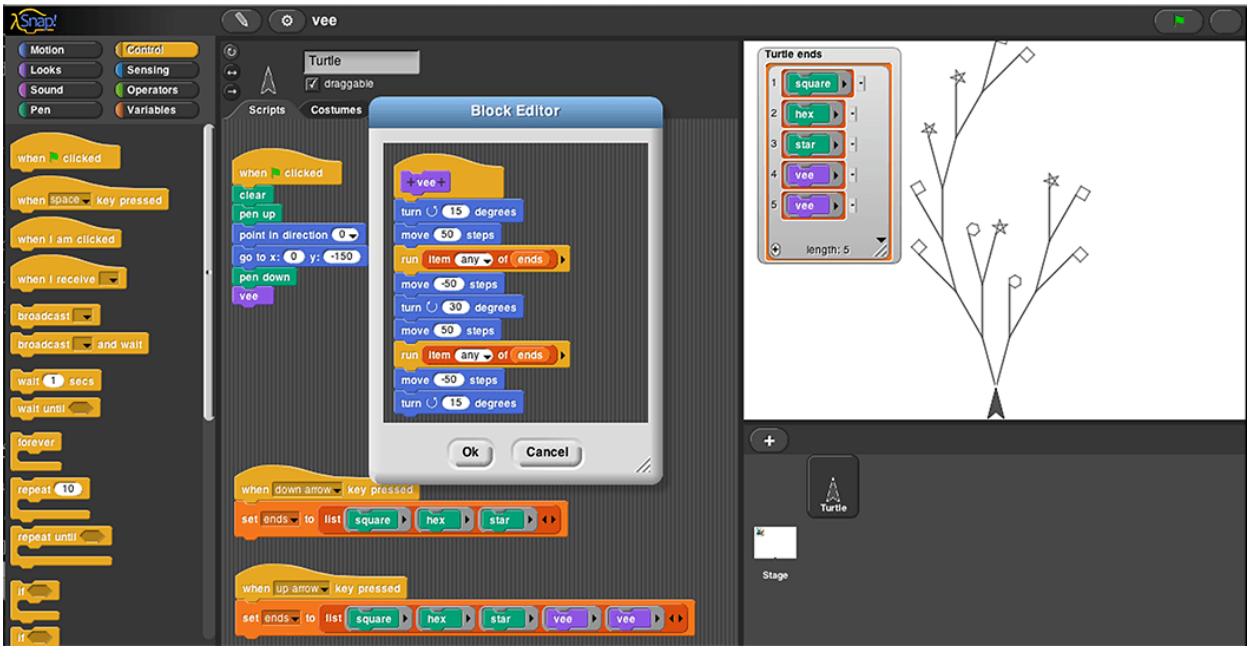


Figura 10: Snap. Imagen descargada de https://snap.berkeley.edu/

## 3.3. Beetle Blocks

Scratch y Snap tienen una gran limitación que es la posibilidad de realizar proyectos en mundos de tres dimensiones. Beetle Blocks [34] soluciona esa carencia y aunque es muy similar a Scratch, incorpora numerosos bloques para poder incorporar una coordenada Z. Con ellos se puede crear cualquier cosa que salga de la imaginación del usuario de forma más realista. Un uso muy útil que se le puede dar es crear el borrador en tres dimensiones de algún objeto que se quiere reproducir en la realidad. En la web oficial podemos encontrar numerosos ejemplos y códigos de otros usuarios para usarlos a modo de guía o para modificarlos y usarlos como punto de partida. No requiere instalación pues el entorno está integrado en su web y es totalmente gratuito.

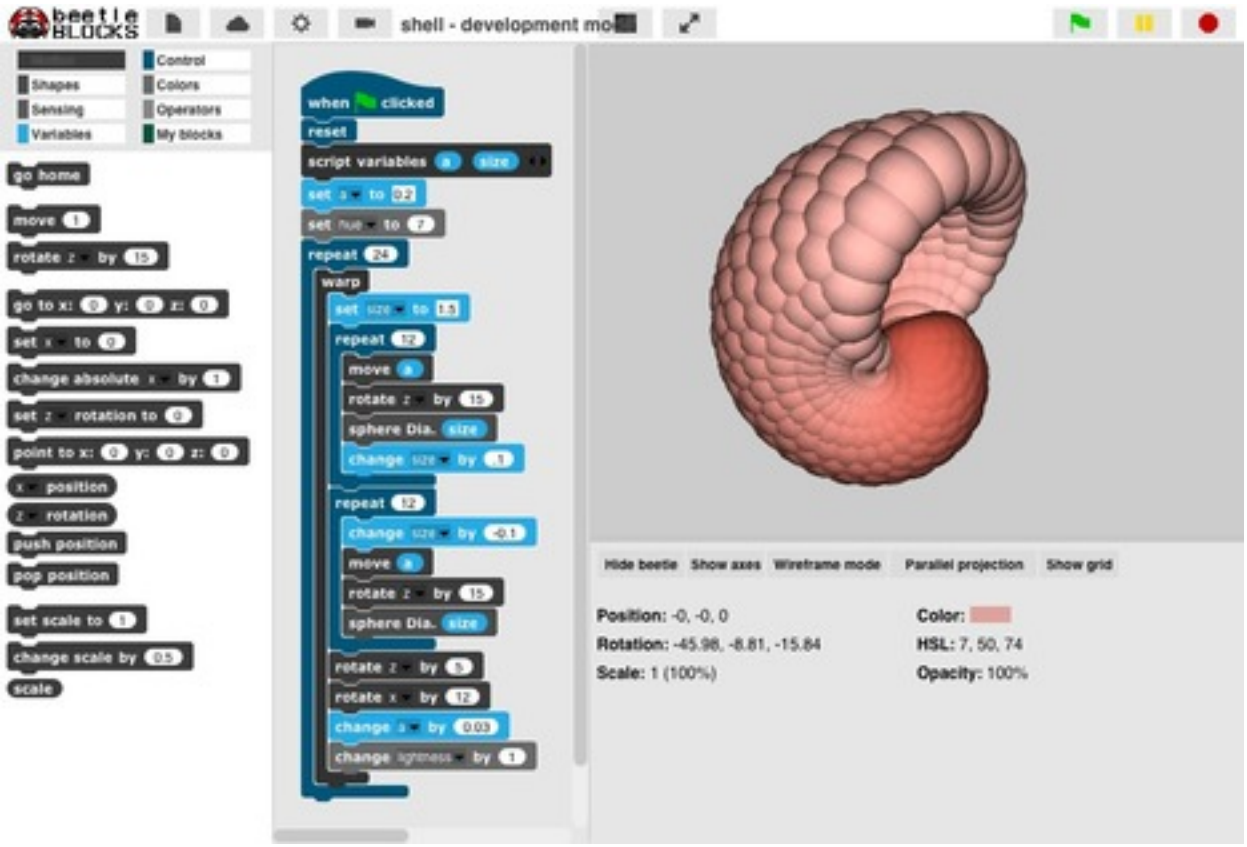


Figura 11: Beetle Blocks. Imagen descargada de scoop.it

## 3.4. Tynker

En el capítulo 2, sección 2.8, hemos hablado de Tynker como web que contiene juegos para aprender a programar. Pero Tynker dispone de un completo editor para materializar cualquier idea que se le ocurra al usuario utilizando bloques visuales. Se pueden crear elementos, agregar sonidos, música y escenarios y realizar interacciones entre ellos mediante los bloques. El mayor problema del editor de Tynker lo encontramos en la exportación pues, aunque se pueden compartir las creaciones, no se pueden exportar de la interfaz de Tynker ni acceder al código interno.

## 3.5. Code.org

También hemos hablado de Code.org en el capítulo 2, sección 2.1. Pero al igual que Tynker, Code.org dispone de un potente editor para poder realizar juegos y aplicaciones mediante bloques visuales. Inicialmente se optó por esta herramienta para realizar los juegos de este Trabajo de Fin de Grado, pero finalmente se comprobó que, aunque los juegos se pueden compartir e introducir en sitios web, aparentemente no se puede acceder al código interno de ellos y entonces no se podrían mover dichos juegos mediante bloques visuales de código.

## 3.6. Google Blockly

Google Blockly [21] [22] [23] es un lenguaje de programación mediante bloques visuales creado por Google. Permite al usuario elaborar código Python, Javascript o PHP mediante los bloques para así no cometer errores de sintaxis. Es la base de multitud de webs mencionadas anteriormente como Code.org, Code For Life o RoboGarden. Cuenta con algunos juegos de ejemplo para solucionar por medio de bloques llamados Blockly Games. Permite al usuario crear cualquier tipo de bloques nuevos para mover con ellos cualquier tipo de código. Tanto el entorno Blockly como los Blockly Games se pueden descargar e implementarlos en otro proyecto combinando sus códigos internos con el código del proyecto. Por ello es la tecnología ideal para poder mover cualquier código ya hecho mediante los bloques y la que se ha elegido para realizar los juegos de este Trabajo de Fin de Grado. Se hablará extensamente de Google Blockly en el capítulo 4.

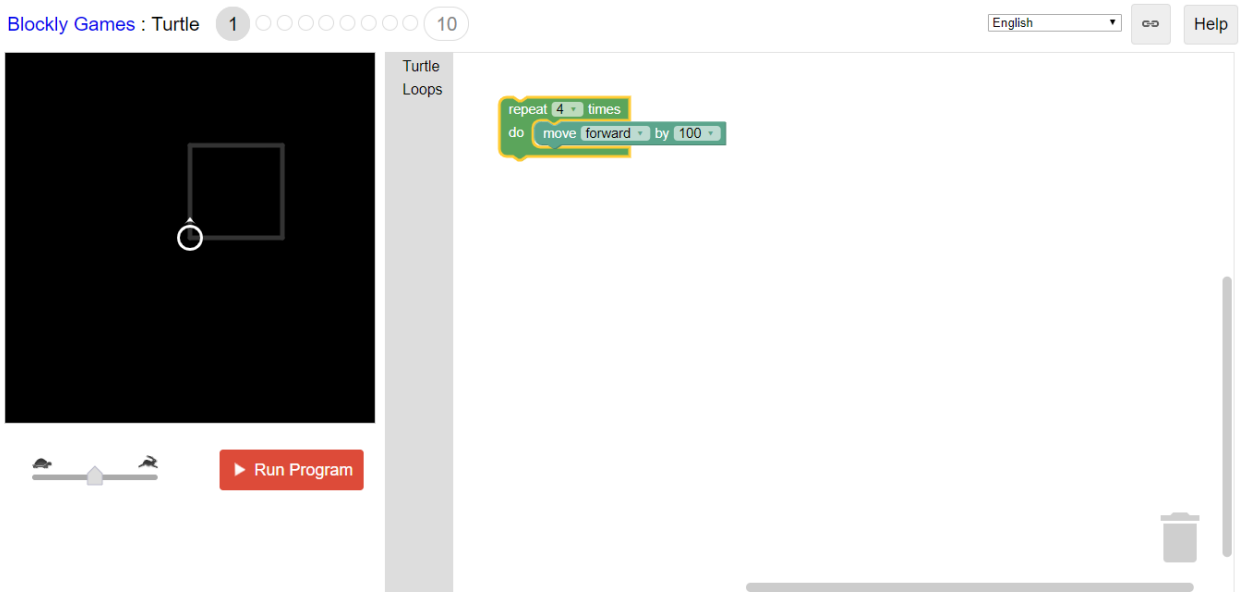


Figura 12: Ejemplo de nivel de Blockly Games. Imagen descargada de Blockly Games

## 3.7. Tabla comparativa. Ventajas y desventajas

En la tabla 2 se van a mostrar las principales ventajas y desventajas de las herramientas que se han presentado. En las filas se representa cada herramienta y en cada columna el nombre, sus ventajas y sus desventajas. En mi opinión me han parecido herramientas muy interesantes pues sirven para crear cualquier cosa que uno pueda imaginar con ayuda de los bloques, dentro de las limitaciones de cada una. Para este proyecto he elegido Google Blockly porque es la herramienta que me permite crear bloques para mover un código externo, en este caso los juegos. No quería realizar el juego y los bloques con la misma herramienta sino unir dos tecnologías diferentes. Me parece una herramienta muy útil pues con ella se podría mover cualquier código de programación existente mediante los bloques y no solamente códigos que componen un juego.

| Nombre | Ventajas | Desventajas |
|---|---|---|
| Code.org | Editor muy completo. Extensos videos y tutoriales para aprender a usar el editor. | No se puede acceder al código interno de las aplicaciones y juegos realizados con el editor. |
| Scratch | Gratuito y software libre. Muy potente. Válido en todos los sistemas operativos. Posibilidad de compartir creaciones en una extensa comunidad. | Limitado a dos dimensiones. |
| Snap | Amplia las funcionalidades de Scratch. Permite crear bloques propios. | Más complejo que Scratch, puede ser más inaccesible para usuarios poco experimentados. Limitado a dos dimensiones |
| Beetle Blocks | Permite crear elementos en tres dimensiones. Numerosos códigos y ejemplos en su web para guiar al usuario. No requiere instalación. | Puede ser difícil crear elementos en tres dimensiones si nunca se utilizó previamente alguna aplicación tipo Scratch. |
| Tynker | Potente editor. Entorno guiado. Contenido de pago. | No se puede acceder al código interno de las aplicaciones y juegos realizados con el editor. |
| Google Blockly | Entorno descargable para implementarlo en cualquier proyecto. Gran comunidad de usuarios y es software libre. Permite crear bloques nuevos | Uso complejo para usuarios poco experimentados en programación. No dispone de un entorno intuitivo. |

Tabla 2: Comparación de Frameworks para crear juegos.

# 4. Google Blockly

Blockly [21] [22] [23] es un lenguaje visual para programar realizado por Google. Se empezó a diseñar a finales del año 2011 y en mayo de 2012 se lanzó una primera versión de prueba. Se compone de diferentes bloques que emulan diferentes sentencias de programación para facilitar la elaboración de código Python, PHP, Dart o Javascript. El usuario no tiene que preocuparse por errores de sintaxis que si aparecerían codificando cualquier lenguaje de programación común. Uno de los puntos fuertes de Google Blockly es la posibilidad de poder crear nuevos bloques y darles la funcionalidad que se necesite. Es un lenguaje en código abierto bajo la licencia Apache License 2.0. y está inspirado en otros entornos de programación con bloques como por ejemplo Scratch. Lo utilizan multitud de proyectos como Code.org, Code For Life, App Inventor o Microsoft MakeCode además de multitud de webs enfocadas al aprendizaje de la programación mediante bloques y/o juegos.

## 4.1. Características

- Software de código abierto: Google Blockly es un lenguaje de código abierto. Permite al usuario utilizarlo e integrarlo en su sitio web o aplicación propios.
- Multilenguaje: Google Blockly ha sido traducido aproximadamente a 50 idiomas.
- Potencia: Google Blockly es capaz de implementar tareas costosas y complejas.
- Exportación: el usuario puede generar sin contratiempos código de programación en el lenguaje deseado a partir de los bloques que haya utilizado.
- Funcionalidad ilimitada: el usuario puede crear nuevos bloques que representen cualquier sentencia de programación en el lenguaje deseado.

## 4.2. Utilizar Blockly en un sitio web

Lo primero que hay que hacer para integrar Google Blockly en un sitio web propio es descargar los archivos fuentes que contienen toda la estructura básica desde el siguiente enlace:
https://github.com/google/blockly/zipball/master
Una vez se han descargado los archivos que contienen todo el material para utilizar Blockly, se incluirán los elementos necesarios en el código HTML. Dentro de la etiqueta HTML de la página incluiremos el Javascript de la siguiente manera:

```
...
<script src="blockly/blockly_compressed.js"></script>
<script src="blockly/blocks_compressed.js"></script>
...
```

Después se deben incluir los mensajes en el idioma preferido ('es' para español, 'en' para inglés...):

```
<script src = "msg/js/es.js"></ script>
```

Para crear la interfaz Blockly hay que añadir una etiqueta 'div' con el tamaño que se quiera dar al panel de bloques:

```
<div id = "blocklyDiv" style = "height: número de px; width:
    número de px;" ></ div>
```

Dentro de esta interfaz hay que crear una *toolBox* o *Caja de herramientas* que contenga los distintos bloques que se pueden utilizar en la aplicación. Se trata de un menu lateral en el que se pueden ver y seleccionar los bloques que queremos utilizar en el panel Blockly. En este ejemplo se mostrará cómo crearlo vacío pues en la siguiente sección se enseñará como crear los bloques y sus tipos utilizando *toolBox*:

```
<xml id = "toolbox" style = "display: none" ></xml>
```

Por último, se inyectará Blockly en nuestra web con esta etiqueta script:

```
<script> var workspacePlayground = Blockly.inject('blocklyDiv', {
    toolbox: document.getElementById('toolbox')}); </script>
```

El resultado será parecido al mostrado en la figura 13.


Figura 13: Entorno Blocky

## 4.3. Toolbox

En el apartado anterior se ha explicado como introducir Blockly en nuestra web, incluyendo una caja de herramientas que contiene todos los bloques que se pueden usar en la aplicación. ¿Pero cómo añadir tipos y bloques en ella? Hay que recordar la etiqueta XML que contiene la *toolBox*:

```
<xml id = "toolbox" style = "␣display:␣none␣" ></xml>
```

Para crear una nueva categoría de bloques que dentro contendrá uno o varios tipos de bloques, utilizar la etiqueta <category>y dentro especificar el nombre que tendrá esa categoría y el número del color con el que se quiere representar:

```
<xml id="toolbox" style="display: none">
<category name="myCategory" colour="110"></category>
</xml>
```

Las categorías de bloques pueden anidarse unas con otras formando árboles de categorías. De esta manera se pueden crear categorías y subcategorías de bloques, aunque su definición será igualmente con la etiqueta <category>.

Para crear un nuevo bloque dentro de la nueva categoría (habrá que tenerlo creado previamente en otro archivo Javascript tal y como se explicará a continuación de este ejemplo) hay que introducir el nombre del tipo en una etiqueta block, todo ello dentro de la etiqueta XML de la *toolBox*:

```
<xml id="toolbox" style="display: none">
<category name="myCategory" colour="110">
<block type="myBlock"></block>
</category>
</xml>
```

Una vez realizado este proceso el nuevo bloque se verá en la *toolbox* como muestra la figura 14.

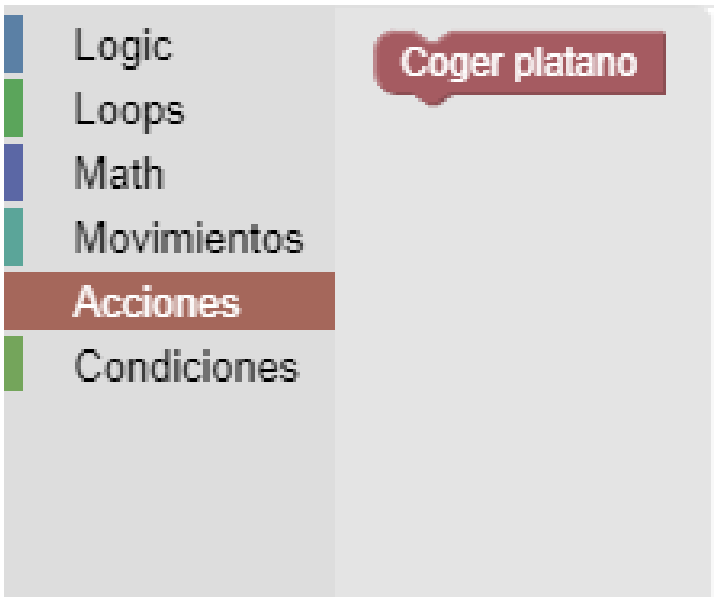


Figura 14: Toolbox. Imagen descargada de: https://developers.google.com/blockly/

## 4.4. Fabricando nuevos bloques

Google Blockly permite crear bloques nuevos propios. Para crear nuevos bloques se necesita crear un nuevo archivo Javascript e incluirlo en el HTML principal. Un bloque está formado por:

- Estructura: define el aspecto visual del bloque como el tipo de bloque o el color.
- Funcionalidad: define que es lo que va a hacer ese bloque, es decir, el código que va a ejecutar.

Para crear nuevos bloques se necesita crear un nuevo archivo Javascript e incluirlo en el HTML principal. Cada bloque está definido por una función que define la parte estructural y una o más funciones que definen la parte funcional. Para realizar estas dos funciones se pueden codificar directamente o utilizar la *Block Factory* que permite diseñar el bloque desde una interfaz que generará el código.

### 4.4.1. Codificación de un nuevo bloque

Para crear la parte visual del bloque desarrollando el código hay que utilizar la función *Blockly.defineBlocksWithJsonArray()* Se debe definir esta función por cada uno de los bloques que se quieran crear. Por ejemplo, si se quieren crear diez bloques, habrá que definir diez funciones *defineBlocks-WithJsonArray* que fabriquen cada uno de ellos. La estructura de esta función es la siguiente:

```
Blockly.defineBlocksWithJsonArray([
{
"type": "myBlock",
"message0": "MYBLOCK",
"previousStatement": true,
"nextStatement": true,
"colour": 25,
"tooltip": "",
"helpUrl": ""
}
]);
```

Algunos de los parámetros que se definen en la función son:

- *Type* es el nombre que se le da al bloque y por el que será identificado dentro de nuestro código para poder operar con él.
- *Message0* es el nombre que aparecerá en la toolbox para ese bloque. Es el nombre que se nos muestra por pantalla cuando se quiere utilizar ese bloque.
- *Colour* definirá el color del bloque.

Ya está creada la parte externa del bloque. Ahora hay que definir la funcionalidad de este bloque. Para ello se utilizará la siguiente función:

```
Blockly.JavaScript['type'] = function(block) {

// Aquí se escribe el código Javascript que ejecutara el bloque.

};
```

Esta función recibirá como parámetro el bloque al que se le quiere otorgar la funcionalidad y para ello le pasaremos como argumento el nombre como se definió previamente en *type*. Una vez realizado todo este proceso, solamente faltaría por agregar el bloque a la *toolbox* tal y como se explicó en el apartado anterior y ya podremos empezar a utilizarlo en el panel de bloques para codificar nuestro propio programa.

### 4.4.2. Bloque a partir de la Block Factory

Otra manera de crear los nuevos bloques es desde un editor de desarrollo web llamado*Blockly Developer Tools*. Esta herramienta se encarga de simplificar algunas tareas de configuración, entre ellas, el desarrollo de nuevos bloques desde la *Block Factory*. De esta forma, mediante una interfaz se podrá definir la estructura y la funcionalidad sin tener que escribir el código manualmente. Se compone de cuatro secciones:

1. Un editor Blockly.
2. Un cuadro de vista donde se puede ver el aspecto visual del bloque creado.
3. Una vista del código que permite generar el bloque. Es el código de la parte estructural.

4. El código resultante de la ejecución de ese bloque. Es el código de la parte funcional.

Tras ponerle un nombre al nuevo bloque se deben utilizar las distintas opciones que hay a la izquierda del editor Blockly para empezar a elaborarlo:

**Colour:** con esta opción se puede elegir entre 360 grados de tonalidad de colores para nuestro bloque.

**Input:** permite asignarle una o más entradas a nuestro bloque. Cada una de ellas debe tener un nombre. También cada bloque puede tener conexiones laterales, superiores o inferiores para interactuar con otros bloques.

**Field:** permiten al usuario agregar campos de entrada al bloque. Por ejemplo, una entrada de texto, de ángulos (de 0 a 360 grados), un campo desplegable con distintas opciones o una variable.

**Type:** asigna un tipo al bloque de manera que solo aceptara entradas del tipo definido. Puede ser Any (cualquier tipo), Boolean, String, Number, Array o se puede crear un nuevo tipo con la opción Other.

Una vez desarrollado el bloque solamente hay que copiar el código de la parte estructural y de la parte funcional en nuestro Javascript y, añadir el nuevo bloque a la *toolbox* de nuestro HTML principal. La interfaz de la Block Factory *Block Factory* se muestra en la figura 15.

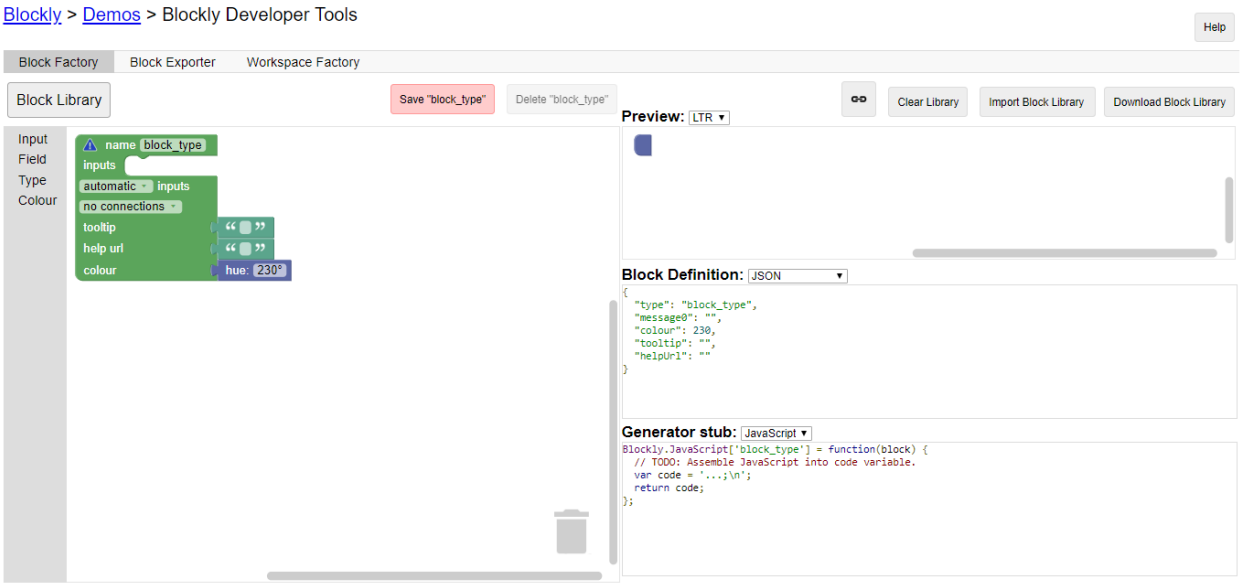


Figura 15: Entorno Blocky

# 5. Frameworks para crear videojuegos con HTML5 y Javascript

Para desarrollar un videojuego existen algunos frameworks que facilitan al usuario la elaboración del mismo. Existen multitud de frameworks para realizar videojuegos en HTML y Javascript. Para poder realizar los juegos de este proyecto necesitaremos utilizar uno de estos frameworks. Por este motivo, en este capítulo se hablará de algunos de los frameworks más conocidos y se analizarán sus ventajas y desventajas con el fin de encontrar el más adecuado para nuestro proyecto. En primer lugar se van a definir algunos conceptos que se utilizarán durante el análisis de estos frameworks:

- Sprite: es una imagen que representa un elemento del juego, ya sea un personaje o un objeto.
- Spritesheet: conjunto de imágenes de un mismo tipo que alternándose entre ellas forman una animación.
- Canvas: es un elemento que nos permite generar gráficos y animaciones por medio de Javascript.
- WebGL: característica de Javascript que mejora las capacidades gráficas que puede mostrar un navegador web.

## 5.1. Panda

Es uno de los frameworks [14] más potentes que existen para fabricar juegos con HTML y Javascript. A finales de 2017 se lanzó una nueva versión final llamada Panda 2. Tiene compatibilidad con todos los sistemas operativos ya sean Windows, Linux y Mac. Cuenta con un sistema de animaciones y de físicas de colisión propio. Trae incorporado un potente editor de código de Javascript donde se puede realizar el código del juego. Este editor facilita la codificación pues cuenta con múltiples características como autocompletado, ejecución en vivo mientras se escribe el código, selección múltiple o personalización a medida del usuario. Además, hay que destacar como una de sus mejores virtudes, su panel de depuración. En él se puede ejecutar el código paso a paso y consultar la información sobre el estado de sus variables y de los elementos que lo componen, así como las interacciones entre ellos y de esta forma poder encontrar posibles errores o comportamientos incorrectos.

También cuenta con una aplicación llamada Panda Remote con la que podremos conectar un dispositivo (ya sea Android, iOS, incluso Xbox One) y así seguir en tiempo real los cambios que se generan en el proyecto.

Es posible incorporar numerosos plugin desde su web oficial y que aumentaran las capacidades de este framework. Por ejemplo, existen plugin que nos permiten hacer juegos para Facebook, añadir un gamepad para poder controlar nuestro juego desde un mando o para agregar físicas y gráficos en tres dimensiones al juego. El gran inconveniente que presenta Panda es la disponibilidad de una versión de pago y una de prueba. La versión de prueba nos permitirá utilizar la mayoría de sus funciones descritas anteriormente, pero si queremos exportar los juegos generados directamente en archivos con extensión IPA y APK para así publicarlos en el store de Android y Apple o utilizar las plantillas de juegos ya completados que están disponibles en la web de Panda, se debe comprar la versión completa. Esta versión completa tiene un precio de 50 dólares al año o 200 dólares si se quiere tener siempre disponible.

## 5.2. Quintus

Quintus [16] [17] es una librería Javascript que permite crear juegos en dos dimensiones mediante lenguaje HMTL para navegadores, teléfonos móviles o incluso televisiones Smart. Tiene su propio sistema de colisiones y físicas. Una de sus principales características es la posibilidad de utilizar herencia entre sus componentes como si de programación orientada a objetos se tratase. Esta característica es útil cuando, por ejemplo, se quiere declarar un nuevo sprite y dotarlo de propiedades que ya posee un sprite ya creado. Esto favorece a la reutilización, ordenación y limpieza del código. Otra gran característica es la posibilidad de crear los niveles en archivos JSON separados y así solamente cargar los niveles que se quieran utilizar fácilmente desde la función de carga. Otro aspecto destacable en Quintus es la utilización de elementos JQuery.

## 5.3. Phaser

Phaser es un framework basado en Pixi.Js que permite desarrollar juegos en HTML5. Es capaz de soportar tanto WebGL como Canvas. La creación de juegos se desarrolla mediante estados que permiten cargar sprites, animaciones y elementos de manera muy simple. A esta sencillez en la carga de

elementos hay que sumar que Phaser es uno de los frameworks más usados pues en la red existe una gran cantidad de documentación, ejemplos y guías. Por estos motivos se ha elegido Phaser como la herramienta adecuada para realizar nuestros juegos de este Trabajo de Fin de Grado. En el capítulo 6 se hablará más extensamente de Phaser y se enseñarán algunas instrucciones básicas para comenzar a desarrollar un juego.

## 5.4. Kiwi

Kiwi [19] es un framework que se utiliza para poder hacer juegos en dos dimensiones para navegadores, sistemas iOS y sistemas Android. Utiliza Canvas y WebGL. Tiene bastantes similitudes con Phaser. Se pueden crear grupos y subgrupos de objetos para poder compartir propiedades y comportamientos de dichos objetos. La forma de añadir al juego elementos como imágenes, sonidos y demás recursos necesarios y su sistema de codificación de estados es muy similar al de Phaser. Los juegos hechos con Kiwi tienen un estado 'Create' donde se declaran las variables y se definen las imágenes, sonidos y demás elementos que se van a utilizar en el juego y un estado 'preload' donde realizan acciones como la carga de esas imágenes en forma de sprites, se les asignan propiedades o se cargan los fondos que van a usar.

## 5.5. Jaws

Inspirada en una librería de Ruby llamada Chingu, Jaws [18] es una librería utilizada para desarrollar juegos en dos dimensiones en lenguaje HTML5. Se puede utilizar como una librería independiente. Como algunos de los otros frameworks vistos en esta misma sección, Jaws utiliza diferentes estados que seccionan algunas funciones del código y el uso de SpriteSheets para realizar animaciones como en Phaser. También comparte con Phaser la facilidad para definir sprites y asignarles teclas para su movimiento. Una de sus principales de ventajas es la posibilidad de utilizar una versión del framework que trae su código en diferentes fragmentos. Esta versión sirve para poder depurar errores pues se carga sección a sección. También dispone de una versión reducida que contiene el Closure Compiler de Google, permitiendo que el código se ejecute de una manera más eficaz y con mayor velocidad. Además es bastante útil para los más principiantes pues si, por ejemplo, al usuario se le olvida

realizar la acción de refresco del juego, Jaws lo refrescará automáticamente.

## 5.6. Melon

Melon [33] es otro framework que se utiliza para crear juegos en dos dimensiones en HTML5. Utiliza WebGL y WebAudio. Es compatible con todos los navegadores existentes y en su propia web se puede comprobar el estado de esa compatibilidad con cada uno de ellos. Se puede utilizar como una librería independiente. Utiliza el algoritmo SAT de colisión basado en polígono para su sistema de colisiones. Incluye funciones exclusivas para facilitar el tratamiento de matrices y vectores. Como se ha visto en otros frameworks también permite agrupar objetos para aplicar propiedades o funciones a la vez a todos ellos y el uso de spritesheets para realizar animaciones. Otra gran ventaja es la posibilidad de integrar mapas en formato JSON o XML. Melon también ofrece todo tipo de compatibilidad con periféricos externos como ratón, teclado e incluso gamepad para poder manejar el juego.

## 5.7. Tabla comparativa. Ventajas y desventajas

A continuación se muestran las tabla 3 y 4 que resumen las ventajas y desventajas de los frameworks presentados. En las filas se representa cada framework y en cada columna el nombre, sus principales ventajas y sus principales desventajas. En mi opinión no hay ningún framework mejor que otro sino que depende de lo que se quiera crear con él. Por eso en algunos no ha sido posible hablar de una desventaja frente a los demás. En este proyecto se escogió Phaser por su desarrollo medante estados y por el gran número de guías y ejemplos que existen en internet. Estas características me parecieron ideales para realizar los juegos de este proyecto con él aunque se podrían haber realizado con alguno de los otros. De hecho, en las conclusiones finales se plantea como una idea de trabajo futuro.

| Nombre | Ventajas | Desventajas |
|---|---|---|
| Panda | Cuenta con un editor completo con funcionalidades como autocompletado o personalización. Posee un panel de depuración. Numerosos plugin que amplían la funcionalidad del framework. | La versión completa con todas las funcionalidades es de pago |
| Quintus | Utiliza propiedades de herencia para poder reutilizar el código. También se pueden crear juegos para móviles y televisores Smart. Posibilidad de crear los niveles en JSON separados. | Más complejo que Scratch, puede ser más inaccesible para usuarios poco experimentados. |
| Phaser | Muy fácil de usar. Instrucciones muy simples. Numerosa documentación, guías y comunidades. Desarrollo del juego por estados bien estructurada. | No se ha encontrado ninguna desventaja notable |
| Kiwi | Sirve también para hacer juegos Android e iOS. Uso de estado. | No se ha encontrado ninguna desventaja notable. |
| Melon | Permite utilizar periféricos. Facilidad para tratar matrices y vectores. | Limitado para realizar juegos más complejos. |

Tabla 3: Comparación de Frameworks para crear juegos.

| Nombre | Ventajas | Desventajas |
|---|---|---|
| Jaws | Uso de estados e instrucciones simples para cargar elementos del juego.<br>Versión del framework en distintas secciones para depurar. | No se ha encontrado ninguna desventaja notable. |
| Melon | Permite utilizar periféricos.<br>Facilidad para tratar matrices y vectores. | Limitado para realizar juegos más complejos. |

Tabla 4: Comparación de Frameworks para crear juegos.

# 6. Phaser

Phaser [31] [32] es un framework de código abierto que permite crear juegos en HTML5. Proporciona una serie de herramientas (carga y animación de sprites, precarga de archivos, sistema de colisiones, movimiento por teclado, etc.) que permiten al usuario agilizar el proceso de desarrollo de un juego. Este desarrollo puede ser para navegadores de escritorio o navegadores móviles. Soporta Canvas y WebGL. Phaser fue ideado por Richard Davey, un apasionado desarrollador de videojuegos de 36 años. Para llevar a cabo el famoso framework, Richard se basó en Pixi.js. Su pasión por este sector y su deseo que de que muchas personas pudieran hacer reales sus ideas le llevaron a realizarlo. Es mantenido por una extensa comunidad.

## 6.1. ¿Dónde encontrar Phaser?

Phaser se puede descargar en su web https://phaser.io/ o en su Github. Una vez descargado el archivo Javascript que contiene el Framework, sólamente hay que añadirlo al HTML propio:

```
< script src= "␣phaser.min.js␣" > </script>
```

A partir de ahí ya es posible utilizar toda la funcionalidad que ofrece y comenzar a crear cualquier juego.

## 6.2. Estados en Phaser

Phaser está basado en una sucesión de estados. Cada estado proporciona una funcionalidad a la aplicación, ya sea la precarga del juego y sus archivos, la inicialización del juego, cada uno de los niveles que componen el juego o el menú inicial. Dentro de cada estado existen una serie de funciones que ayudan a realizar las tareas básicas necesarias. Algunos ejemplos de funciones son:

**Init:** estado inicial del juego. Es el primer estado en cargar al iniciarse el juego.

**Preload:** en este estado se cargan los recursos necesarios del juego como las imágenes de los sprites para los personajes, los fondos de pantalla o el audio.

**Create:** en este estado se crean los elementos que se van a utilizar en el juego y que previamente se han cargado en el estado preload.

**Update:** contiene toda la lógica del juego. Este estado se ejecuta con cada cambio que ocurre en el juego.

Podemos crear cuantas funciones queramos dentro de nuestro estado siempre que lo necesitemos. Para crear un estado hay que crear una variable que contenga todas las funciones que necesitamos para ofrecerle la funcionalidad deseada:

```
var Estado = {
init: function ( ) { ... }
preload: function ( ) { ... }
}
```

Para que un estado de un juego se ejecute hay que agregarlo a la lista de estados e inicializarlo cuando se quiera utilizar:

```
myGame.state.add('estado', loadState);
myGame.state.start('estado');
```

Es preciso tener el código estructurado en funciones para que esté lo más ordenado posible. Para cargar los archivos necesarios para el juego es necesario tener una función que se encargue de ello. A continuación, se muestra como cargar una imagen para un juego:

```
var bootState = {
preload: function () {
this.myGame.load.image('nombreImagen', 'ubicacionImagen');
}
```

## 6.3. Sprites estáticos y animados

En Phaser existen dos tipos principales de imágenes que representan los elementos del juego: sprites y spritesheets.

Necesitaremos una o más imágenes que serán las protagonistas de nuestro juego. Los sprites son cada una de los imagenes que van a aparecer en nuestro juego. Para crear un sprite primero se necesita realizar un preload de la imagen con la que se quiere representar:

```
function preload(){
juego.load.sprite('nombreParaElSprite','ubicacionImagen');
}
```

Un spritesheet es un archivo que contiene varios sprites más reducidos en su interior. Se utiliza normalmente para representar un personaje en movimiento o un determinado número de objetos. La principal dificultad que se tiene que tener qué en cuenta para cargar un spritesheet es que hay que comprobar las medidas exactas de cada una de las pequeñas imágenes que lo componen pues Phaser las almacenará en una especie de vector de spirtes. Para cargar un spritesheet se utilizará la siguiente instrucción:

```
function preload() {
this.game.load.spritesheet(nombreImagen, ubicacionImagen,
    AltoDeLaParticion, AnchoDeLaParticion, Nú
    meroDeImagenesEnElArchivo );
}
```

La diferencia fundamental entre la carga de un sprite y un spritesheet es que este último necesita tres parámetros adicionales como son el alto y el ancho de cada una de las imágenes que lo componen, y el número de imágenes totales para poder dividirlas de manera correcta.

Una vez cargada nuestra imagen se debe crear una variable para el sprite o spritesheet utilizando la imagen deseada y las coordenadas donde se quiere posicionar. Para mantener una cierta lógica esta operación la realizaremos en la función create:

```
this.mySprite = this.game.add.sprite(coordenadax, coordenaday, '
    nombreImagen');
```

Existen múltiples operaciones que se pueden realizar sobre los sprites o spritesheets. Algunas son:

- Cambiar coordenada x: mySprite.x = número;
- Cambiar coordenada y: mySprite.y = número;
- Cambiar el alto de la imagen: mySprite.width = número;
- Cambiar el ancho de la imagen: mySprite.height = número;
- Cambiar el tamaño a escala de la imagen: mySprite.scale.setTo(número);

- Crear un grupo de multiples sprites para poder operar sobre ellos simultaneamente: mySprite = myGame.add.group()
- Hacer visible un elemento: mySprite.alpha = 1;
- Hacer invisible un elemento: mySprite.alpha = 0;
- Mover un sprite mediante una animación: mySprite.body.moveTo(duración, distancia, dirección(UP,DOWN,LEFT,RIGHT));

En el caso de los spritesheets vamos a destacar dos operaciones básicas. Al estar formados por un vector de spirtes podemos acceder a cada uno ellos por separado (Por ejemplo, para acceder al sprite almacenado en la tercera posición usaríamos sprite.frame = 2) o bien asociarlos. Esta asociación puede ser útil para animar un sprite. Para ello podemos agrupar los sprites según el movimiento que queramos mostrar:

```
sprite.animations.add("nombre", [rango de imagenes], velocidad,
    booleano para repetir o no la animacion).
```

## 6.4. Sistema de Colisiones

Lo más normal es que los elementos de un juego interactúen entre sí. Un personaje puede desplazarse sobre un suelo o chocarse contra una pared, enemigo u objeto. Es por ello que necesitamos un sistema de colisiones. A priori puede ser lo más complicado al elaborar un videojuego, pero afortunadamente, una de las virtudes de Phaser son sus sistemas de colisiones.

Hay varias maneras de definir un sistema de colisiones en Phaser por ello se va a explicar el más sencillo de todos. Si tenemos dos sprites definidos en la función preload y creados en la función create, en primer lugar debemos definir el sistema de colisión arcade en el juego con physics.startSystem (Phaser.Physics.ARCADE). Una vez hecho esto activamos la física en cada uno de los sprites con game.physics.enable(sprite, Phaser.Physics.ARCADE). Después de eso solo nos queda comprobar la colisión en la función update. Un código de ejemplo para una colisión entre dos sprites sería:

```
game = new Phaser.Game(400, 400, Phaser.AUTO, 'ejemplo',{preload:
    preload,create:create,update:update});
var game,
sprite1, sprite2;
```

```
function preload(){
game.load.image("sprite1", "sprite1.png");
game.load.image("sprite2", "sprite2.png");
};

function create() {
game.physics.startSystem(Phaser.Physics.ARCADE);
s1 = game.add.sprite(32, 100, 'sprite1');
s2 = game.add.sprite(60, 100, 'sprite2');
game.physics.enable(s1, Phaser.Physics.ARCADE);
game.physics.enable(s2, Phaser.Physics.ARCADE);
};

function update() {
game.physics.arcade.collide(alien, grupoDiamante, colision, null,
    this);
};
```

## 6.5. Música y sonidos

Si se quiere añadir música o sonidos al juego se asigna a una variable la carga del sonido y después se reproduce con el comando play:

```
game.load.audio('MySong', 'music.mp3');
song = this.game.add.audio('MySong');
song.play();
```

Puede haber problemas de políticas en Google Chrome al cargar archivos de audio en un proyecto Phaser.

## 6.6. Juego realizado con Phaser: Peceras

A modo de ejemplo, en esta sección se va a explicar cómo se ha elaborado uno de los juegos para este proyecto. Peceras es un juego que consta de cuatro peceras. Una de las peceras es la mayor de todas y contiene siete peces en su interior. Encima de ella tenemos otras tres peceras, dos de un tamaño mediano y una de un tamaño menor. Los detalles de este juego se explicarán en el capítulo 8, sección 8.4. Se ha elegido este juego para la explicación por su sencillez gráfica. A continuación, se van a explicar los pasos y los detalles para generar la parte de visual de este juego con Phaser.

En primer lugar, hay que crear un nuevo archivo Javascript donde se empezará a desarrollar el código del juego. En primer lugar se creará una función *window.onload* y dentro de ella estará todo el código del juego. Esta función se encarga de lanzar un evento cuando todo el contenido se ha cargado y en este caso, iniciará el juego una vez se hayan cargado sus elementos. Dentro de esta función declararemos una variable que contenga el juego de la siguiente manera:

```
window.onload = function () {
var juego = new Phaser.Game(800, 600, Phaser.CANVAS, 'juego', {
    preload: preload, create: create, update: update});
};
```

Le asignamos a la variable juego un nuevo elemento *Phaser.Game* y llevará como parámetros el largo y ancho de la pantalla que contendrá el juego, Phaser.CANVAS para poder generar los gráficos, un parámetro que indique el nombre del juego y por último, los estados que componen el juego para que sean cargados. No hay que olvidar que los juegos en Phaser están formados por diferentes estados. Si no se indican al declarar la variable inicial, no se cargarán.

El siguiente paso consiste en declarar los diferentes estados. Vamos a crear un estado *preload* que cargará los elementos que se vayan a utilizar en el juego, el estado *create* que creará esos elementos dentro del juego y el estado *update* que se ejecutará constantemente. En el estado *preload* vamos a cargar los sprites de los peces y las peceras, el fondo y la música del juego.

```
function preload() {
juego.load.spritesheet('peces','sprites/peces.png', 48, 48 , 96);
juego.load.image('pecera', 'sprites/pecera.png');
juego.load.image('arena', 'sprites/arena.jpg');
juego.load.audio('song', 'sprites/fishsong.wav');
}
```

El archivo del que se cargarán los peces viene en forma de *spritesheets* para facilitar su animación, por lo que hay que cargarlo como tal pasándole como parámetros el alto y ancho de cada imagen y, el número de imagenes totales del archivo.

El siguiente estado a desarrollar y el más extenso es el *create*. En este estado vamos a crear los distintos elementos del juego a partir de las cargas

realizadas en el estado *preload*. También declararemos las variables que vamos a utilizar en el juego. En primer lugar, se creará el fondo de pantalla a partir de la imagen *arena*. Crearemos una variable para mostrar texto que por ahora no contendrá nada y que más tarde actualizaremos en *update* y crearemos cuatro variables que indicarán el número de peces de cada pecera. A continuación, se crearán las cuatro peceras indicando la posición en la que se quiere mostrar cada una y se cambiará su tamaño a escala con *setTo*. Para crear los peces se ha optado por crear peces en todas las peceras y solamente hacer visibles los de la pecera grande. En primer lugar, se cargan los siete peces de la pecera grande, se le cambia el tamaño a escala y se crea una animación con las tres primeras imágenes del archivo *peces.png* que cargamos en *preload*. Con *animations.add* definimos el nombre de la animación, las imágenes del archivo cargado que queremos utilizar y la velocidad de movimiento para cada pez. Con *animations.play* reproducimos dicha animación para cada pez. En los peces posicionados en las otras peceras pondremos la propiedad *alpha* a cero para hacerlos invisibles y jugaremos con esta propiedad para mover los peces de una pecera a otra. Lo último que haremos en este estado será añadir la música y reproducirla.

```
function create() {
juego.add.tileSprite(0, 0, 800, 600, 'arena');
text = juego.add.text(158, 10);
numpeces = 7;
numpeces1 = 0;
numpeces2 = 0;
numpeces3 = 0;
peceragrande = this.add.sprite(215, 275, 'pecera');
peceragrande.scale.setTo(0.45);
pecera1 = this.add.sprite(20, 45, 'pecera');
pecera1.scale.setTo(0.25);
pecera2 = this.add.sprite(330, 100, 'pecera');
pecera2.scale.setTo(0.17);
pecera3 = this.add.sprite(575, 45, 'pecera');
pecera3.scale.setTo(0.25);
pez1 = this.add.sprite(250, 415, 'peces');
pez1.scale.setTo(1.3);
pez1.animations.add('pezazul',[0,1,2], 10, true);
pez1.animations.play('pezazul');
pez2 = this.add.sprite(300, 415, 'peces');
pez2.scale.setTo(1.3);
pez2.animations.add('pezazul',[0,1,2], 10, true);
pez2.animations.play('pezazul');
```

```
pez3 = this.add.sprite(350, 415, 'peces');
pez3.scale.setTo(1.3);
pez3.animations.add('pezazul',[0,1,2], 10, true);
pez3.animations.play('pezazul');
pez4 = this.add.sprite(400, 415, 'peces');
pez4.scale.setTo(1.3);
pez4.animations.add('pezazul',[0,1,2], 10, true);
pez4.animations.play('pezazul');
pez5 = this.add.sprite(450, 415, 'peces');
pez5.scale.setTo(1.3);
pez5.animations.add('pezazul',[0,1,2], 10, true);
pez5.animations.play('pezazul');
pez6 = this.add.sprite(500, 415, 'peces');
pez6.scale.setTo(1.3);
pez6.animations.add('pezazul',[0,1,2], 10, true);
pez6.animations.play('pezazul');
pez7 = this.add.sprite(375, 475, 'peces');
pez7.scale.setTo(1.3);
pez7.animations.add('pezazul',[0,1,2], 10, true);
pez7.animations.play('pezazul');
pez11 = this.add.sprite(40, 120, 'peces');
pez11.scale.setTo(1.3);
pez11.animations.add('pezazul',[0,1,2], 10, true);
pez11.animations.play('pezazul');
pez11.alpha = 0;
pez12 = this.add.sprite(80, 120, 'peces');
pez12.scale.setTo(1.3);
pez12.animations.add('pezazul',[0,1,2], 10, true);
pez12.animations.play('pezazul');
pez12.alpha = 0;

pez13 = this.add.sprite(120, 120, 'peces');
pez13.scale.setTo(1.3);
pez13.animations.add('pezazul',[0,1,2], 10, true);
pez13.animations.play('pezazul');
pez13.alpha = 0;
pez21 = this.add.sprite(365, 140, 'peces');
pez21.scale.setTo(1.3);
pez21.animations.add('pezazul',[0,1,2], 10, true);
pez21.animations.play('pezazul');
pez21.alpha = 0;

pez31 = this.add.sprite(600, 120, 'peces');
pez31.scale.setTo(1.3);
pez31.animations.add('pezazul',[0,1,2], 10, true);
```

```
pez31.animations.play('pezazul');
pez31.alpha = 0;
pez32 = this.add.sprite(640, 120, 'peces');
pez32.scale.setTo(1.3);
pez32.animations.add('pezazul',[0,1,2], 10, true);
pez32.animations.play('pezazul');
pez32.alpha = 0;

pez33 = this.add.sprite(680, 120, 'peces');
pez33.scale.setTo(1.3);
pez33.animations.add('pezazul',[0,1,2], 10, true);
pez33.animations.play('pezazul');
pez33.alpha = 0;
music = juego.add.audio('song');
music.play();
}
```

La última función que se necesita para terminar la parte visual es *create*. Está función se ejecutará constantemente y se utilizará para verificar la condición para ganar el juego que es que los peces de las peceras superiores estén visibles, es decir, que tengan su propiedad *alpha* a uno y los peces de la pecera inferior tengan su propiedad *alpha* a cero. Si es así la variable text que se ha creado en el anterior estado mostrará *Juego superado.*

```
function update() {
if(pez11.alpha == 1 && pez12.alpha == 1 && pez13.alpha == 1 &&
    pez21.alpha == 1 && pez31.alpha == 1 && pez32.alpha == 1 &&
    pez33.alpha == 1 ) {
text.setText("Juego superado");

}
}
```

Utilizando *Phaser* se ha podido crear la parte visual del juego. Existen multitud de instrucciones en *Phaser* que nos permitirán realizar juegos más completos y complejos.

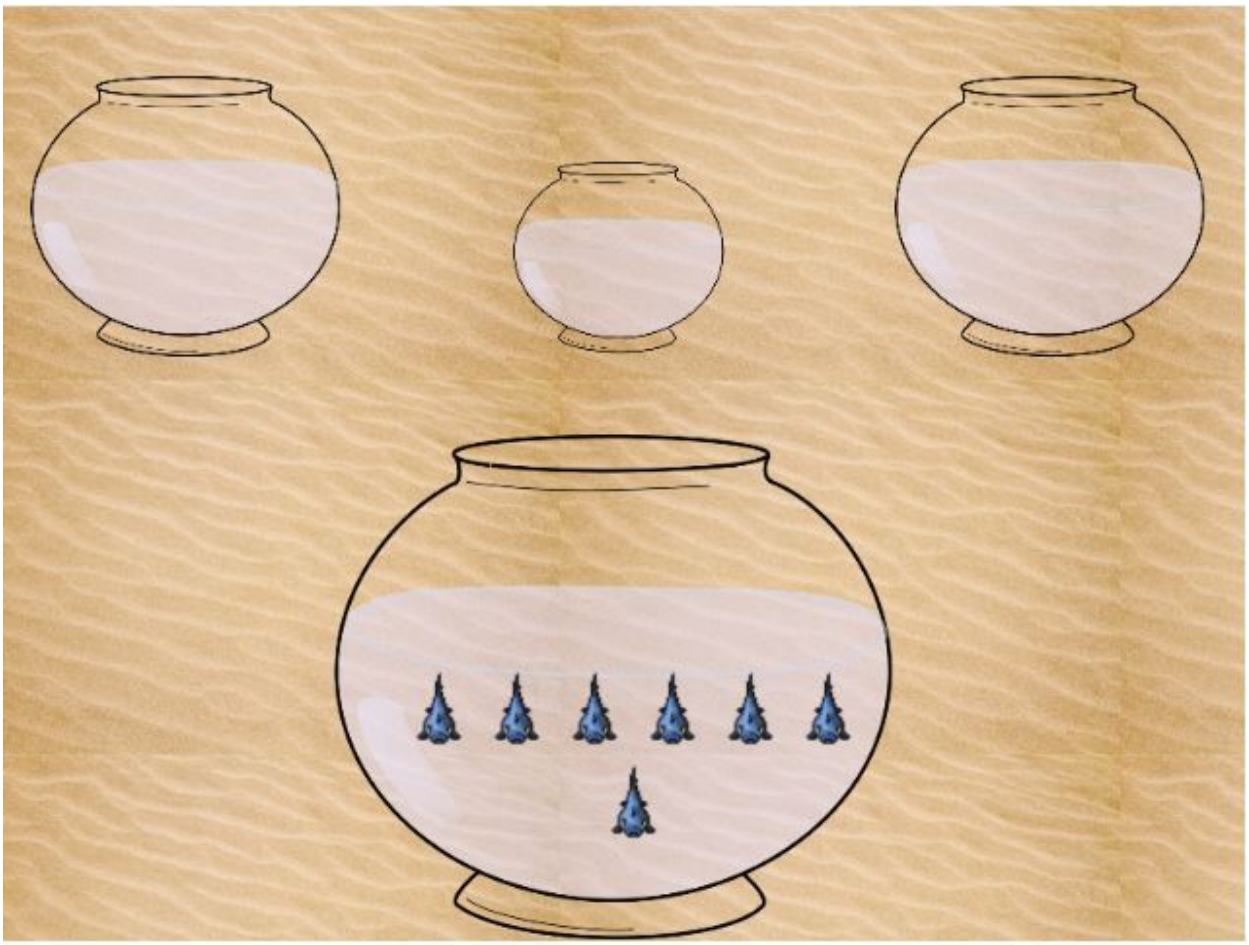

Figura 16: Juego creado para este trabajo: Peceras

# 7. Combinando Google Blockly y Phaser

Una vez se ha visto cómo utilizar Google Blockly y como desarrollar juegos con el framework Phaser, se necesita realizar una interacción mutua entre ambos elementos. En este capítulo se explica cómo combinar Blockly con un juego realizado con Phaser y como hacer que los bloques muevan el código del juego.

Una vez tenemos nuestro juego desarrollado con Phaser, es hora de añadirle los bloques que lo moverán. En primer lugar, hay que crear un archivo HTML que contenga tanto el juego realizado con Phaser, como los archivos de Google Blockly tal y como se muestra a continuación:

```
<script src="Phaser.js"> </script>
<script src="blockly/acorn_interpreter.js"> </script>
<script src="blockly/myinterpreter.html"> </script>
<script src="blockly/blockly_compressed.js"> </script>
<script src="blockly/blocks_compressed.js"> </script>
<script src="blockly/wait_block.js"> </script>
<script src="blockly/javascript_compressed.js"> </script>
<script src="blockly/bloqueNuevo.js"> </script>
<script src="blockly/msg/js/en.js"> </script>
<script src="JuegoPhaser.js"> </script>
```

A continuación, hay que insertar la ToolBox tal y como se explicó en la sección **??** del capítulo 4. Una vez hecho esto y tras ordenar los elementos y darles el tamaño adecuado, ya tendríamos en nuestro HTML tanto el panel de bloques como el juego desarrollado en Phaser. También se necesitan las funciones que hagan funcionar el Interpreter para ir ejecutando cada uno de los bloques. Estas funciones se pueden mirar en la web de Blockly *https://developers.google.com/blockly/guides/app-integration/running-javascript*, que contiene un ejemplo del que se puede extraer todo el código del panel de bloques. Para llamar al *Interpreter* hay que utilizar la siguiente función:

```
var code = Blockly.JavaScript.workspaceToCode(workspace);
var myInterpreter = new Interpreter(code);
myInterpreter.run();
```

También necesitamos la función de la llamada a la API a la que se llamará cada vez que haya un bloque en el panel que no se ha ejecutado aún:

```
function initApi(interpreter, scope) {
```

```
// Add an API function for the alert() block.
var wrapper = function(text) {
return alert(arguments.length ? text : '');
};
interpreter.setProperty(scope, 'alert',
interpreter.createNativeFunction(wrapper));

// Add an API function for the prompt() block.
wrapper = function(text) {
return prompt(text);
};
interpreter.setProperty(scope, 'prompt',
interpreter.createNativeFunction(wrapper));
}
```

El siguiente paso va a ser crear un nuevo archivo Javascript que contendrá los bloques nuevos que se quieran crear y añadirlo a nuestro HTML:

```
<script src="MisBloques.js"> </script>
```

En este archivo vamos a desarrollar todos los bloques nuevos que van a ser utilizados en nuestro juego. Tal y como explicamos en el la sección **??** del capítulo 4, para crear un bloque primero hay que definir su estructura con la función *Blockly.defineBlocksWithJsonArray*. Para definir el código que ejecutará ese bloque necesitamos dos funciones más. Esta división en dos funciones es útil si se quieren utilizar los nuevos bloques dentro de bucles de programación. Llegados a este punto vamos a diferenciar entre dos tipos de bloques: bloques condicionales y bloques de código, que presentaremos a continuación.

## 7.1. Bloques de código

Los bloques de código son los que contienen instrucciones que modifican alguna parte del programa, en este caso de nuestro juego. En primer lugar, definiremos una primera función que llevará como parámetro el nombre del bloque y que solamente devolverá una llamada a la segunda función:

```
Blockly.JavaScript['NombreBloque'] = function (block) {

return "nombreSegundaFuncion();\n"

}
```

A continuación, creamos la segunda función con la siguiente estructura y definiendo el código que va a ejecutar el bloque en el lugar correspondiente:

```
function initInterpreterNombreDeLaFuncion (interpreter, scope) {
Blockly.JavaScript.addReservedWords('nombreSegundaFuncion');
var wrapper = interpreter.createAsyncFunction(
function (callback) {
setTimeout(
function () {

// Codigo que va a ejecutar el bloque

callback();
},1000
);
});
interpreter.setProperty(scope, 'nombreSegundaFuncion', wrapper);
}
```

Ahora que tenemos las dos funciones definidas tenemos que agregar el nombre de esta última función en el *initApi* del *Interpreter* de Blockly:

```
function initApi(interpreter, scope) {
// Add an API function for the alert() block.
var wrapper = function(text) {
return alert(arguments.length ? text : '');
};
interpreter.setProperty(scope, 'alert',
interpreter.createNativeFunction(wrapper));

// Add an API function for the prompt() block.
wrapper = function(text) {
return prompt(text);
};
interpreter.setProperty(scope, 'prompt',
interpreter.createNativeFunction(wrapper));
initInterpreterNombreDeLaFuncion(interpreter, scope)
```

Por último, solamente queda crear el nuevo bloque en la *toolbox* . Se puede meter dentro de una categoría existente o crear una nueva:

```
<block type="nombreDelBloque"> </block>
```

## 7.2. Bloques condicionales

Los bloques condicionales son los que se unen a los bloques que representan instrucciones de control para definir su condición de parada. Normalmente hay que definirlos como booleanos para que evalúen si la condición es o no correcta. Para definir la parte visual hay que añadir un campo *output* que tenga valor *boolean*. Al igual que hicimos con los bloques de código, necesitamos definir dos funciones, pero no de la misma manera. La primera función sería:

```
Blockly.JavaScript['NombreDelBloque'] = function (block) {
var code = 'NombreDeLaSegundaFuncion() ';
return [code, Blockly.JavaScript.ORDER_MEMBER];
};
```

La diferencia principal es que para los bloques condicionales debemos devolver a la segunda función una estructura que además de la llamada a la función devuelva Blockly.Javascript.ORDER-MEMBER . La segunda función también varía ligeramente respecto al bloque de código:

```
function initInterpreterNombreDeLaFuncion(interpreter, scope) {
Blockly.JavaScript.addReservedWords('nombreSegundaFuncion');
var wrapper = interpreter.createNativeFunction(
function () {
return (Condicion a evaluar. Un ejemplo sería if (x > 3))
});
interpreter.setProperty(scope, 'nombreSegundaFuncion', wrapper);
}
```

En este caso no se requiere la llamada a callback como ocurría en el bloque de código ya que directamente se devuelve la condición a evaluar. Por último, al igual que en el otro tipo de bloques, quedaría definir la función en el *Interpreter* y crear el bloque en la *toolbox*. Con estas funciones los bloques nuevos ya están listos para usarse y ejecutarse con éxito. Si queremos acceder a un elemento del juego desarrollado en Phaser, solamente tenemos que definir el código correspondiente en uno de los bloques nuevos y dicho bloque modificará el comportamiento de ese elemento.

## 7.3. Creando bloques nuevos para un juego realizado en Phaser: Peceras

A continuación, se van a ver los pasos explicados en este capítulo aplicados a un juego real que se ha realizado para este proyecto. En la sección 6.6 del capítulo 6 se explicó cómo realizar la parte visual del juego *Peceras*. Ahora para que el juego funcione debemos crear bloques nuevos que muevan los peces de una pecera a otra. Realmente lo que se hará es hacer invisible al pez que queremos mover y hacer visible al pez de la posición destino donde queremos moverlo. Para ello habrá que cambiar los valores *alpha* de cada pez a cero o a uno según se quiera hacer invisible o visible. En primer lugar, se va a explicar cómo crear un bloque de código que permita mover los peces de la pecera grande a la primera pecera y después se explicará cómo realizar un bloque condicional siguendo el proceso de codificación de bloques. Se necesita un archivo Javascript nuevo donde irán los nuevos bloques. Para crear el bloque de código hay que definir su parte visual con la siguiente función:

```
Blockly.defineBlocksWithJsonArray([
{
"type": "grandeauno",
"message0": "Mover pez de la pecera grande a la primera pecera",
"previousStatement": null,
"nextStatement": null,
"colour": 170,

}
]);
```

Como se explicó en el capítulo 4, sección 4.4.1 para crear la parte visual de un bloque se requiere la función *Blockly.defineBlocksWithJsonArray*. El argumento *type* indica el identificador del bloque y el argumento *message0* indica el mensaje que contendrá el bloque y que se verá en el panel de bloques. Después se indica que no es necesario que lleven un bloque previo encajado a él ni un argumento y se indica el color del bloque. Con esta función ya está definida la parte visual del bloque. Como hemos visto en la sección 7.1 de este capítulo, para definir la parte funcional necesitamos dos funciones que llamaremos *Blockly.JavaScript['grandeauno']* y *initInterpretergrandeauno*. La primera de estas funciones solamente devuelve un valor de retorno que llama a la palabra reservada de la segunda función:

```
Blockly.JavaScript['grandeauno'] = function (block) {

return "grandea1();\n";

};
```

La función *initInterpretergrandeauno* implementará el código que va a ejecutar el nuevo bloque. Este bloque lo que tendrá que hacer es hacer invisible al pez que se va a mover desde la pecera grande y hacer visible al pez de la pecera uno. Para ello habrá que comprobar cuantos peces hay en la pecera grande y cuantos hay en la pecera uno para así saber cuáles son los peces cuya propiedad *alpha* hay que cambiar:

```
function initInterpretergrandeauno(interpreter, scope) {

Blockly.JavaScript.addReservedWords('grandea1');
var wrapper = interpreter.createAsyncFunction(
function (callback) {
setTimeout(
function () {
if(numpeces == 6) {
if(numpeces1 == 0) {
pez6.alpha = 0;
pez11.alpha = 1;
numpeces--;
numpeces1++;
} else if(numpeces1 == 1) {
pez6.alpha = 0;
pez12.alpha = 1;
numpeces--;
numpeces1++;
} else if(numpeces1 == 2) {
pez5.alpha = 0;
pez13.alpha = 1;
numpeces--;
numpeces1++;
}
} else if(numpeces == 5) {
if(numpeces1 == 0) {
pez5.alpha = 0;
pez11.alpha = 1;
numpeces--;
numpeces1++;
} else if(numpeces1 == 1) {
pez5.alpha = 0;
```

```
pez12.alpha = 1;
numpeces--;
numpeces1++;
} else if(numpeces1 == 2) {
pez5.alpha = 0;
pez13.alpha = 1;
numpeces--;
numpeces1++;
}
} else if(numpeces == 4) {
if(numpeces1 == 0) {
pez4.alpha = 0;
pez11.alpha = 1;
numpeces--;
numpeces1++;
} else if(numpeces1 == 1) {
pez4.alpha = 0;
pez12.alpha = 1;
numpeces--;
numpeces1++;
} else if(numpeces1 == 2) {
pez5.alpha = 0;
pez13.alpha = 1;
numpeces--;
numpeces1++;
}
} else if(numpeces == 3) {
if(numpeces1 == 0) {
pez3.alpha = 0;
pez11.alpha = 1;
numpeces--;
numpeces1++;
} else if(numpeces1 == 1) {
pez3.alpha = 0;
pez12.alpha = 1;
numpeces--;
numpeces1++;
} else if(numpeces1 == 2) {
pez3.alpha = 0;
pez13.alpha = 1;
numpeces--;
numpeces3++;
}
} else if(numpeces == 2) {
if(numpeces1 == 0) {
```

```
pez2.alpha = 0;
pez11.alpha = 1;
numpeces--;
numpeces1++;
} else if(numpeces1 == 1) {
pez2.alpha = 0;
pez12.alpha = 1;
numpeces--;
numpeces1++;
} else if(numpeces1 == 2) {
pez5.alpha = 0;
pez13.alpha = 1;
numpeces--;
numpeces1++;
}
} else if(numpeces == 1) {
if(numpeces1 == 0) {
pez1.alpha = 0;
pez11.alpha = 1;
numpeces--;
numpeces1++;
} else if(numpeces1 == 1) {
pez1.alpha = 0;
pez12.alpha = 1;
numpeces--;
numpeces1++;
} else if(numpeces1 == 2) {
pez5.alpha = 0;
pez13.alpha = 1;
numpeces--;
numpeces1++;
}
} else if(numpeces == 7) {
if(numpeces1 == 0) {
pez7.alpha = 0;
pez11.alpha = 1;
numpeces--;
numpeces1++;
} else if(numpeces1 == 1) {
pez7.alpha = 0;
pez12.alpha = 1;
numpeces--;
numpeces1++;
} else if(numpeces1 == 2) {
pez5.alpha = 0;
```

```
pez13.alpha = 1;
numpeces--;
numpeces1++;
}
}
callback();
},1000
);
});
interpreter.setProperty(scope, 'grandea1', wrapper);
}
```

Los últimos pasos a seguir en nuestro HTML principal son añadir el bloque nuevo a la *toolbox* mediante el identificador que se le asigno al definir la parte visual y, añadir la cabecera de la función *initInterpretergrandeauno* al *Interpreter* de Blockly. Una vez hecho nuestro nuevo bloque está listo para funcionar.

Ahora se va a explicar cómo crear un bucle condicional. Se quiere desarrollar un nuevo bloque que unido a una estructura de control nos diga si quedan más peces en la pecera principal. Al igual que con los bloques de código hay que crear la parte visual y la parte funcional:

```
Blockly.defineBlocksWithJsonArray([
{
"type": "haypecesgrande",
"message0": "Hay␣peces␣en␣la␣pecera␣grande",
"output": "Boolean",

"previousStatement": null,
"nextStatement": null,
"colour": 270,

}
]);
```

En la parte visual hay una pequeña diferencia con la que se definía en los bloques de código y es que hace falta asignar el *output* a tipo *boolean* para poder combinarlos con los bloques de estructuras de control a modo de condición. Para la parte funcional se vuelven a definir dos funciones, pero con algunos cambios respecto a los bloques de código:

```
Blockly.JavaScript['haypecesgrande'] = function (block) {
```

```
var code = 'pecesgrande() ';
return [code, Blockly.JavaScript.ORDER_MEMBER];

};
```

Esta vez no se devuelve la llamada a la función solamente sino que necesita de la llamada *Blockly.Javascript.ORDER-MEMBER* u otras similares (en este proyecto se ha utilizado esta pues, no es relevante para nuestros propósitos) que indica el orden de precedencia de algunas operaciones. Queda por definir la segunda función que implementa el código del bloque, pero al ser un bloque condicional será ligeramente distinta a la del bloque de código:

```
function initInterpreterhaypecesgrande(interpreter, scope) {

Blockly.JavaScript.addReservedWords('pecesgrande');
var wrapper = interpreter.createNativeFunction(
function () {
return numpeces>0;
});
interpreter.setProperty(scope, 'pecesgrande', wrapper);
}
```

En esta función hay que devolver la condición que se quiere evaluar. En este caso queremos saber si hay peces en la pecera grande, número que se representa con la variable *numpeces*. Por último habría que definir en el HTML principal el bloque en la *toolbox* y la cabecera de la función *initInterpreter-haypecesgrande* en el *Interpreter* de Blockly. Una vez terminados estos pasos, ya se podría utilizar el nuevo bloque condicional uniéndolo a los bloques de estructuras de control tal y como se muestra en la figura 17.

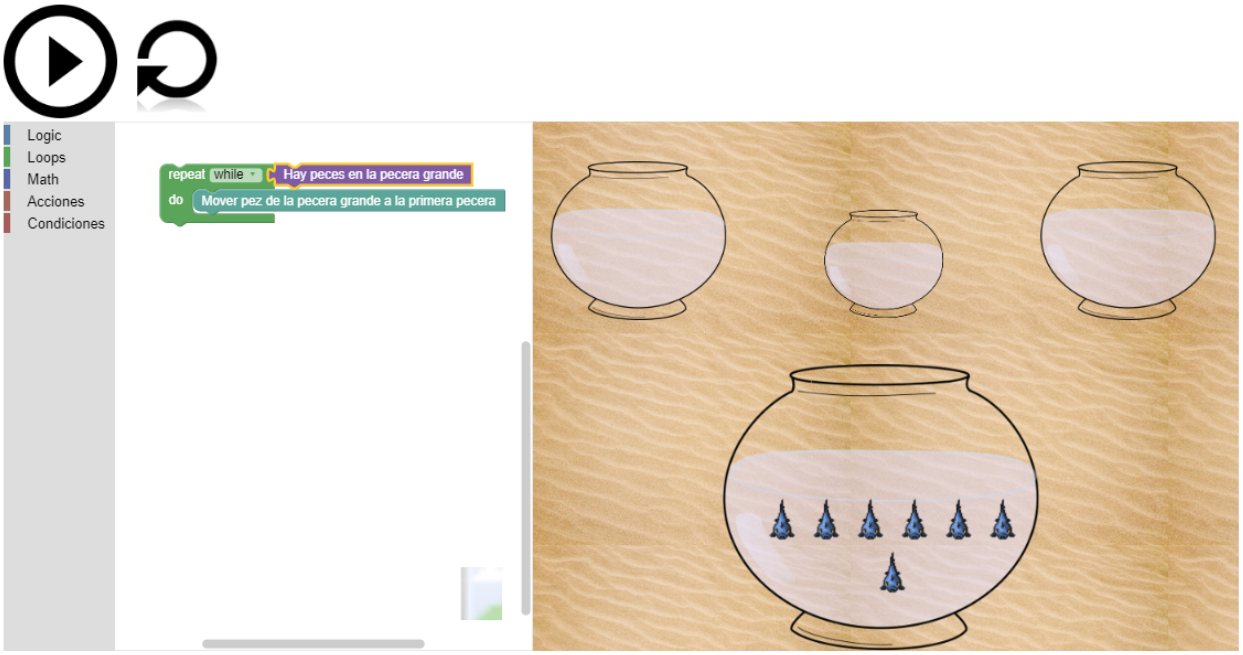


Figura 17: Juegos Peceras y los nuevos bloques creados

# 8. Juegos realizados para este proyecto

En este capítulo se van a presentar los cuatro juegos que se han realizado para este proyecto. Para cada juego se explicará la temática, las instrucciones de uso, los conocimientos didácticos que se quieren enseñar y las validaciones realizadas para evitar fallos. Los juegos están enlazados mediante un menú principal con cuatro botones que lleva a cada uno de ellos. Están desarrollados mediante el framework Phaser y contienen un panel de bloques. Este panel de bloques incluye bloques de instrucciones de control y algunos bloques creados exclusivamente para cada juego, y se utilizan para codficar las soluciones. Con estos juegos se ha querido reflejar todo lo aprendido durante la realización de este proyecto y con intención de que sirvan para enseñar algunos conceptos didácticos de programación. Además, se pretende que estos juegos sean parte del inicio de futuros proyectos basados en Blockly. El código de cada uno de estos juegos se puede descargar desde el siguiente enlace: `https://drive.google.com/open?id=1Pr925P4Uy-jafces_CsDq99IPzB-3hEr`.

## 8.1. Recolectando plátanos

En este juego se debe guiar a nuestro protagonista, un hambriento mono que quiere subir por un platanero en busca de comida. El platanero tiene en sus ramas unos apetitosos plátanos, la comida favorita de nuestro mono. Nuestro protagonista quiere recolectar todos los plátanos que tiene el árbol ¿Será capaz el mono de hacerse con todos los plátanos y disfrutar de un buen festín para saciar toda su hambre?

### 8.1.1. Instrucciones del juego

El jugador tiene que guiar al personaje hacia los diferentes plátanos que se encuentran en el árbol utilizando los bloques visuales de código. Los bloques disponibles son escalara tronco, trepar rama izquierda, trepar rama derecha, volver a la raíz y coger plátano. Escalar tronco sirve para subir el tronco del árbol. Una vez allí habrá que moverse por las ramas utilizando trepar rama izquierda o trepar rama derecha. Cuando el personaje se encuentre en una rama con un plátano, se podrá utilizar coger plátano para obtener un punto que subirá al marcador. Cuando el jugador haya recogido todos los plátanos que se encuentran en las ramas y baje del árbol, habrá completado el juego.

### 8.1.2. Conocimientos didácticos que se desean transmitir

Este juego simula un recorrido para el árbol binario que se representa en el juego. El árbol platanero representa un árbol binario, el mono es el encargado de recorrer dicho árbol y la existencia o no de plátanos en las ramas representa el valor de los nodos. Los movimientos de trepar rama izquierda o trepar rama derecha son similares a las funciones del árbol binario para visitar el hijo izquierdo o el hijo derecho. La acción de coger un plátano sería equivalente a obtener el valor del nodo actual y la acción de volver a la raíz simula la vuelta después de las llamadas recursivas para recorrer este árbol binario. Cada una de las instrucciones se ejecutará para el nodo actual donde se encuentra el personaje. La codificación de la solución está ideada para que el movimiento del mono pueda simular las llamadas recursivas para el árbol representado en el juego, visitar el nodo destino y regresar al nodo raíz.

### 8.1.3. Validaciones realizadas

En este apartado se han realizado una serie de validaciones para comprobar posibles errores que puedan ocurrir durante el juego. Todos los casos que se exponen han sido comprobados para garantizar el buen funcionamiento del juego.

- El personaje no se desplaza fuera del árbol o a ramas que no existen.

- No se suman puntos si se intenta recolectar un plátano en una rama que no hay nada y tampoco se suman puntos si se intenta recolectar dos veces el mismo plátano.

- Todos los movimientos que permiten volver a la raíz, ir a la rama izquierda, ir a la rama derecha se realizan correctamente.

- Se gana la partida cuando se han recolectado todos los plátanos quedando aún movimientos o habiéndolos gastado todos.


Figura 18: Juego creado para este trabajo: Recolectando plátanos

## 8.2. Viaje en coche

Juego basado en el problema "Viaje en coche"de la asignatura Diseño de Algoritmos del Grado en Ingeniería de Computadores de la Universidad Complutense de Madrid.
Un viajante tiene que viajar en coche de un punto a otra siguiendo una ruta preestablecida. Con el depósito lleno su coche puede recorrer un máximo de kilómetros y por el camino existen varias gasolineras en las que el viajante puede parar para llenar su depósito. El viajante quiere parar el menor número de veces en su camino por lo que tiene que elegir en que gasolineras debe parar y en cuáles no. ¿Será capaz de encontrar una solución óptima a su problema?

### 8.2.1. Conocimientos didácticos que se desean transmitir

Este juego presenta un problema cuya solución se puede obtener mediante un método voraz. El jugador debe elegir al encontrarse una gasolinera, si debe parar o no para rellenar el depósito para así encontrar la solución en la que menos veces tenga que parar el coche, es decir, la solución óptima. Sólo se dará como ganador al jugador cuando haya encontrado una solución óptima.

### 8.2.2. Instrucciones del juego

El jugador puede mover el coche con los bloques de código. El coche comienza con el depósito vacío de gasolina. Cuando el coche esté enfrente de una gasolinera, se podrá utilizar el bloque *llenar depósito* para rellenar completamente el depósito de gasolina del coche. Si se queda sin gasolina no podrá seguir avanzando y habrá que comenzar desde el principio. Solamente cuando se consiga la solución óptima que incluya el menor número de paradas, se dará por superado el juego. El número entre gasolineras indica la distancia que hay entre el espacio de las dos gasolineras.

### 8.2.3. Validaciones realizadas

En este apartado se han realizado una serie de validaciones para comprobar posibles errores que puedan ocurrir durante el juego. Todos los casos que se exponen han sido comprobados para garantizar el buen funcionamiento del juego.

- El coche no puede repostar cuando no hay una gasolinera.
- El coche no puede moverse si no tiene gasolina.
- Se gana la partida cuando se encuentra la solución con el menor número de paradas.
- Se pierde la partida cuando no se encuentra la solución óptima.


Figura 19: Juego creado para este trabajo: Viaje en coche

## 8.3. Noche de Navidad

Por fin es Navidad y como cada año Santa Claus debe repartir los regalos por toda la ciudad. Una tarea tan agotadora que ha provocado que en la última ciudad que visitó olvidó dejar regalos en todas las casas. ¿Serás capaz de utilizar los bucles e instrucciones de programación para que los habitantes de esas casas no se queden sin regalo de Navidad?

### 8.3.1. Conocimientos didácticos que se desean transmitir

Para solucionar este juego de la manera más completa posible hay que utilizar bloques de instrucciones de control y condicionales pues si se deja un regalo en una casa que ya tenía uno, no se podrá completar el juego. Por ello se recomienda comprobar en que casa hay un regalo y en cual no para que el algoritmo quede lo más completo posible. De esta manera el jugador podrá reforzar o demostrar conocimientos utilizando los bloques *while*, *if* y condicionales.

### 8.3.2. Instrucciones del juego

El jugador debe recoger los regalos del trineo (Santa únicamente puede cargar con dos regalos, pero podría volver a por más en caso de ser necesario) y moverse por cada una de las casas con las instrucciones de movimiento. Solamente se debe dejar el regalo en las casas que no lo tienen pues si se deja en una de las casas que lo tienen, habrá que volver al trineo a recoger otro regalo. El juego termina cuando todas las casas tienen un regalo y Santa Claus no lleve ninguno encima (En caso contrario hay que dejarlos en el trineo).

### 8.3.3. Validaciones realizadas

En este apartado se han realizado una serie de validaciones para comprobar posibles errores que puedan ocurrir durante el juego. Todos los casos que se exponen han sido comprobados para garantizar el buen funcionamiento del juego.

- El personaje no puede moverse a una zona donde no haya un edificio.
- No se puede dejar un regalo en una casa si el personaje no lleva ningún regalo.
- Sólo se pueden llevar encima dos regalos.
- Al dejar un regalo en una casa el número de regalos decrece en uno.

- La partida termina si todas las casas tienen regalo y Santa no lleva ninguno encima.


Figura 20: Juego creado para este trabajo: Noche de Navidad

## 8.4. Peceras

Una tienda de animales ha recibido siete maravillosos ejemplares de peces azules en una gran pecera. Para tenerlos más visibles por toda la tienda, el dueño quiere repartirlos en tres peceras: dos peceras contendrán tres peces cada uno y la otra pecera contendrá sólamente un pez. ¿Podrías ser capaz de repartir correctamente los peces en cada pecera?

### 8.4.1. Conocimientos didácticos que se desean transmitir

Para solucionar este juego de la manera más completa posible hay que utilizar bloques de instrucciones de control y condicionales. La idea del juego es que además de resolverlo, el jugador sea capaz de realizar bucles y condicionales para ir moviendo los peces de un lugar a otro. Además de adquirir habilidad utilizando las instrucciones de control *if* y *while*, también se tiene en cuenta problemas de capacidad que se pueden dar en el mundo de la programación real con algunas estructuras de datos.

### 8.4.2. Instrucciones del juego

El jugador debe mover los peces desde la pecera grande a las otras tres peceras. En la primera pecera caben tres peces, en la segunda cabe un pez y en la tercera caben tres peces. Se recomienda al jugador que además de resolver el juego, practique la realización de bucles y condicionales pasando los peces de la pecera grande a la primera pecera, y después cada uno de los peces pase por la segunda pecera y por la tercera pecera.

### 8.4.3. Validaciones realizadas

En este apartado se han realizado una serie de validaciones para comprobar posibles errores que puedan ocurrir durante el juego. Todos los casos que se exponen han sido comprobados para garantizar el buen funcionamiento del juego.

- No se puede sobrepasar el máximo de peces de cada pecera.

- Al pasar un pez de una pecera a otra, éste pasa a estar solamente en la pecera destino.

- El juego termina cuando las tres peceras están llenas y la pecera grande está vacía.

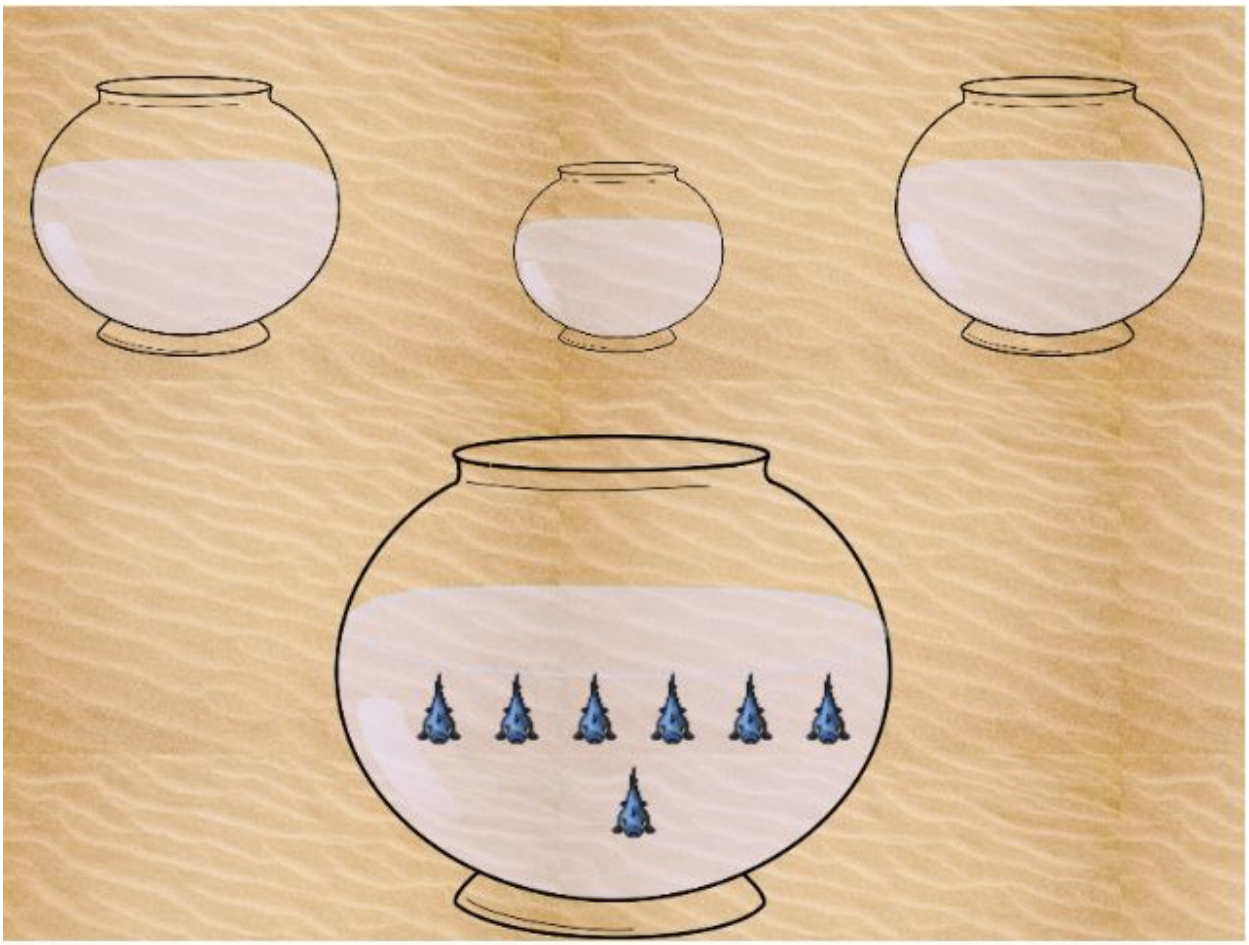

Figura 21: Juego creado para este trabajo: Peceras

# 9. Conclusiones

En este capítulo se va a analizar los objetivos que se han alcanzado con la realización de este trabajo. También se analizarán los puntos a mejorar, las dificultades que se han encontrado durante la realización del trabajo y que trabajo futuro se podría plantear a partir de este proyecto.

Los primeros objetivos consistían en realizar diferentes estudios sobre las webs que contienen juegos para programar, las herramientas que utilizan bloques visuales de programación y los frameworks que se utilizan para crear videojuegos, y realizar una comparativa entre ellos. Estos estudios han sido recogidos en esta memoria, pero podrían mejorarse con una comparación más profunda entre ellos. Esta comparativa ha servido para poder seleccionar los elementos más adecuados para realizar los juegos. Se ha escogido Phaser como framework para realizar los juegos y Google Blockly para implementar el sistema de bloques. Se ha elegido Google Blockly por su sencilla implementación en una web HTML y por la facilidad que tiene para crear bloques propios que sean capaz de mover un código de programación. Por otro lado, se ha elegido Phaser como framework para diseñar los juegos debido a su sencillez para cargar los elementos, por su división en estados y por las numerosas guías, ejemplos y documentación que se puede encontrar sobre este framework en la red. Una vez se han seleccionado las herramientas ha habido que aprender cómo utilizar cada una de ellas. En el caso de Phaser ha habido que aprender cómo realizar un juego en HTML y Javascript con ayuda del framework. Además, se planteaba como objetivo que la información recogida en esta memoria sirviera como guía para ayudar a desarrollar nuevos juegos. Este objetivo se ha cumplido al enseñar cómo crear los elementos básicos y los estados para realizar un juego, pero podría haberse profundizado en aspectos más avanzados. En el caso de Blockly ha habido que aprender a implementarlo en una web, a crear nuevos bloques y como hacer que interactúe con el HTML y Javascript del juego, objetivo que se ha cumplido, aunque se podría mejorar la explicación de cómo juntar ambas tecnologías pues puede resultar difícil de entender si no se tiene algún conocimiento previo sobre el tema. Una vez se han obtenido los conocimientos necesarios para llevar a cabo el objetivo final de nuestro proyecto, se han realizado cuatro juegos en HTML y Javascript que se solucionan mediante los bloques de Google Blockly y algunos otros bloques nuevos creados para cada juego. Aunque el desarrollo de los juegos con las dos tecnologías ha podido realizarse, considero que no se

ha conseguido ese objetivo didáctivo en todos ellos.

Una vez realizados se ha comprobado que se pueden realizar juegos de este tipo juntando varias tecnologías. Uniendo Phaser y Blockly se podrían realizar muchos de los juegos que hemos visto en cada una de las webs. Se ha logrado realizar una herramienta con la que se puede transmitir conocimientos informáticos a los jugadores.

Durante la realización de este proyecto se han encontrado algunas dificultades. La compatibilidad en todos los navegadores ha sido un problema frecuente. El panel de bloques se ve oscuro en Internet Explorer mientras que en Google Chrome se ve perfectamente. Por otro lado, el audio no siempre se escucha en Google Chrome mientras que en Internet Explorer carga perfectamente. Esto es debido a un problema de políticas CORS del navegador de Google y se puede solucionar usando el protocolo Http o instalando ciertas extensiones de Chrome. También en Chrome se observa que a veces no funciona la papelera para eliminar bloques del panel de Blockly. En su lugar se puede eliminar haciendo clic derecho sobre el bloque y seleccionando la opción correspondiente. Otro problema que ha surgido al realizar este proyecto está relacionado con las animaciones de los juegos. Para mover un sprite de un lugar a otro y que no se mueva bruscamente si no con una animación desde el punto origen hasta el punto destino, hay que utilizar una instrucción de Phaser llamada body.moveTo (Ya hablamos de está instrucción en el capítulo dedicado a Phaser). Al utilizar está instrucción se pierden las coordenadas X, Y del sprite. Aunque se puede saber en qué posición está el sprite, si queremos acceder a las coordenadas aparecerán como nulas. Este hecho ha provocado que los juegos que se han realizado para este proyecto no tengan transiciones de movimiento, pues todos se han basado en posiciones y coordenadas para los eventos que ocurren en ellos.

Considero que este trabajo construye una base con la cual se pueden crear algunas ideas interesantes que quedan pendientes. Una buena idea sería realizar ejercicios de asignaturas impartidas en la Facultad de Informática de la Universidad Complutense de Madrid como pueden ser Estructura de Datos y Algoritmos o Diseño de Algoritmos. Otra propuesta sería realizar estos mismos juegos u otros nuevos utilizando otra herramienta diferente

a Google Blockly y otro framework diferente a Phaser. De esta manera se podrían comprobar con más detalle las diferencias y limitaciones de cada herramienta para realizar el mismo tipo de juego. Una última propuesta que me encantaría realizar en un futuro sería implementar Google Blockly en aplicaciones software hechas con distintos lenguajes de programación y usar los bloques visuales para mover su código.

# Conclusions

In this chapter we will analyze the objectives that have been achieved with the realization of this work. The points to be improved will also be analyzed, the difficulties that have been encountered during the realization of the work and what future work could be proposed from this project.

The first objectives consisted in carrying out different studies on the websites that contain games to program, the tools that use visual blocks and the frameworks that are used to create videogames, and make a comparison between them. These studies have been collected in this report, but could be improved with a deeper comparison between them. This comparison has served to select the most appropriate elements to make the games. Phaser has been chosen as a framework to make the games and Google Blockly to implement the block system. Google Blockly has been chosen for its simple implementation in an HTML web and for the ease it has to create own blocks that are capable of moving a programming code. On the other hand, Phaser has been chosen as a framework to design the games due to its simplicity to load the elements, its division into states and the numerous guides, examples and documentation that can be found on this framework on the network. Once the tools have been selected, it has been necessary to learn how to use each of them. In the case of Phaser, we had to learn how to make a game in HTML and Javascript with the help of the framework. In addition, it was proposed that the information collected in this report serve as a guide to help develop new games. This objective has been fulfilled by teaching how to create the basic elements and the states to make a game, but it could have been deepened in more advanced aspects. In the case of Blockly, it has been necessary to learn how to implement it on a website, create new blocks and how to interact with the game's HTML and Javascript, an objective that has been met, although the explanation of how to bring both technologies together could be improved. It can be difficult to understand if you do not have some prior knowledge on the subject. Once the necessary knowledge has been obtained to carry out the final objective of our project, four games have been made in HTML and Javascript that are solved by the blocks of Google Blockly and some other new blocks created for each game. Although the development of the games with the two technologies has been possible, I believe that this teaching objective has not been achieved in all of them.

Once made, it has been verified that games of this type can be made by combining several technologies. Uniting Phaser and Blockly could be done many of the games we have seen in each of the websites. It has been possible to create a tool with which computer knowledge can be transmitted to the players.

During the realization of this project some difficulties have been encountered. Compatibility in all browsers has been a frequent problem. The block panel looks dark in Internet Explorer while in Google Chrome it looks perfectly. On the other hand, audio is not always heard in Google Chrome while in Internet Explorer loads perfectly. This is due to a CORS policy problem in the Google browser and can be fixed using the Http protocol or by installing certain Chrome extensions. Also in Chrome it is observed that sometimes the trash can not remove blocks from the Blockly panel. Instead it can be deleted by right clicking on the block and selecting the corresponding option. Another problem that has arisen when carrying out this project is related to the animations of the games. To move a sprite from one place to another and not to move abruptly if not with an animation from the origin point to the destination point, we must use a Phaser instruction called body.moveTo (We already talked about this instruction in the Phaser chapter). When using this instruction, the X, Y coordinates of the sprite are lost. Although you can know in what position the sprite is, if we want to access the coordinates they will appear as null. This fact has caused that the games that have been made for this project do not have transitions of movement, since all have been based on positions and coordinates for the events that occur in them.

I believe that this work builds a base with which you can create some interesting ideas that remain pending. A good idea would be to carry out algorithm exercises of subjects taught in the Universidad Complutense de Madrid. Another proposal would be to make these same or new games using another tool different from Google Blockly and another framework different from Phaser. This way you could check in more detail the differences and limitations of each tool to perform the same type of game. One last proposal that I would love to do in the future would be to implement Google Blockly in software applications made with different programming languages and use the visual blocks to move their code.

# Bibliografía

# Referencias

## Información para la memoria


[1] Otras herramientas para aprender a programar *https://www.educaciontrespuntocero.com/recursos/programacion/herramientas-online-aprender-programar/35063.html*

[2] CodeMonkey *https://www.playcodemonkey.com/*

[3] Snap *https://www.programoergosum.com/blog/25-aprende-a-programar-con-snap*

[4] Snap *https://programamos.es/snap-una-herramienta-muy-potente-de-programacion-visual/*

[5] Scratch *https://es.wikipedia.org/wiki/scratch*

[6] Scratch *https://garajeimagina.com/es/blog/2016/08/22/que-es-scratch-y-para-que-sirve/*

[7] Kodu *https://kodu.softonic.com/*

[8] Kodu *http://recursostic.educacion.es/observatorio/web/eu/software/software-general/779-kodu-aprendiendo-a-programar-nuestros-propios-juegos*

[9] LaTeX *http://nokyotsu.com/latex/figuras.html*

[10] TeXworks *http://www.tug.org/texworks/*

[11] TeXworks *https://beebom.com/best-latex-editors/*

[12] Brackets *http://brackets.io/*

[13] Brackets *https://damiandeluca.com.ar/principales-ventajas-de-utilizar-brackets-como-editor-de-codigo-para-html-css-y-javascript*

[14] Panda *https://www.panda2.io/*

[15] Payground *http://playgroundjs.com/*

[16] Quintus *http://www.html5quintus.com/*

[17] Quintus *https://www.udemy.com/juegos-moviles-html5/*

[18] Jaws *http://jawsjs.com/*

[19] Kiwi *http://www.kiwijs.org/*

[20] Programación con bloques *https://www.madewithcode.com/*

[21] Google Blockly *https://developers.google.com/blockly/*

[22] Google Blockly *https://en.wikipedia.org/wiki/Blockly*

[23] Google Blockly *https://www.genbetadev.com/herramientas/google-blockly-un-lenguaje-visual-para-aprender-a-programar*

[24] Educacion 3.0 *https://www.educaciontrespuntocero.com/recursos/programacion/lenguajes-programacion-informatica-para-primaria-secundaria/32011.html*

[25] Code.org *https://code.org/*

[26] Robogarden *https://robogarden.ca/*

[27] Code For Life *https://www.codeforlife.education/*

[28] Codecombat *https://codecombat.com/*

[29] Tynker *https://www.tynker.com/*

[30] CheckIO *https://checkio.org/*

[31] Phaser *https://phaser.io/*

[32] Phaser *http://www.phaser-hispano.gq/p/la-gran-guia-de-phaserjs-en-espanol.html*

[33] Melon *http://melonjs.org/*

[34] Beetle Blocks *http://beetleblocks.com/*

## Elementos utilizados para los juegos


[35] Música de los juegos *https://freesound.org/*

[36] Sprite mono *https://forums.rpgmakerweb.com/index.php?threads/whtdragons-animals-and-running-horses-now-with-more-dragons.53552/*

[37] Sprite árbol *http://worldartsme.com/images/tree-trunk-and-branches-clipart-1.jpg*

[38] Sprite plátano *https://www.codeandweb.com/spriteilluminator*

[39] Fondo Recolectando plátanos *https://www.pinterest.es/pin/214343263492100794/*

[40] Sprite coche *https://dribbble.com/shots/4503772-Desert-Road*

[41] Fondo juego Viaje en coche *https://www.pinterest.es/pin/214343263492100794/*

[42] Sprite gasolinera *https://www.iconfinder.com/icons/2443117/*

[43] Sprite casa *https://pixabay.com/es/casa-iconos-tarjeta-jims-1294564/*

[44] Icono Reset y Play *https://icons.webtoolhub.com/icon-n21490-detail.aspx*

[45] Sprite Santa Claus *https://www.gameart2d.com/santa-claus-free-sprites.html*

[46] Sprite Regalo *https://image.flaticon.com/icons/png/128/1244/1244248.png*

[47] Sprite Casa de Navidad *http://icons.iconarchive.com/icons/iconshow/construction/256/H icon.png*

[48] Sprite Trineo *https://vignette.wikia.nocookie.net/clubpenguin/images/f/f9/Santa27sSleigh.*

[49] Sprite Pecera *https://pixabay.com/es/fishbowl-vacC3ADo-el-agua-vidrio-304200/*

[50] Sprite Peces *https://forums.rpgmakerweb.com/index.php?threads/lizard-sprite.74091/*

[51] Botones Menú *http://www.buttongenerator.com/result.php*

[52] Imagen Menú *https://www.videoblocks.com/video/retro-80s-vhs-tape-video-game-intro-landscape-vector-arcade-wireframe-terrain-4k-hf0yxps4ipdkrsug*

[53] Fuente de texto Arcade Menú *https://www.dafont.com/es/arcade-classic-pizz.font*